

---

# **RIPE Atlas Sagan Documentation**

*Release 1.2.2*

**Daniel Quinn**

**Aug 30, 2017**



---

## Contents

---

|          |                                       |          |
|----------|---------------------------------------|----------|
| <b>1</b> | <b>Why This Exists</b>                | <b>3</b> |
| 1.1      | Requirements & Installation . . . . . | 3        |
| 1.2      | Use & Examples . . . . .              | 4        |
| 1.3      | Attributes & Methods . . . . .        | 8        |
| 1.4      | How To Contribute . . . . .           | 19       |
| 1.5      | Changelog . . . . .                   | 20       |



A parsing library for RIPE Atlas measurement results



---

## Why This Exists

---

RIPE Atlas generates a **lot** of data, and the format of that data changes over time. Often you want to do something simple like fetch the median RTT for each measurement result between date *X* and date *Y*. Unfortunately, there are dozens of edge cases to account for while parsing the JSON, like the format of errors and firmware upgrades that changed the format entirely.

To make this easier for our users (and for ourselves), we wrote an easy to use parser that's smart enough to figure out the best course of action for each result, and return to you a useful, native Python object.

Contents:

## Requirements & Installation

### Requirements

As you might have guessed, with all of the magic going on under the hood, there are a few dependencies:

- `cryptography`
- `python-dateutil`
- `pytz`

Additionally, we recommend that you also install `ujson` as it will speed up the JSON-decoding step considerably, and `sphinx` if you intend to build the documentation files for offline use.

### Installation

Installation should be easy, though it may take a while to install all of the aforementioned requirements. Using `pip` is the recommended method.

### Using pip

The quickest and easiest way to install Sagan is to use `pip`:

```
$ pip install ripe.atlas.sagan
```

### From GitHub

If you're feeling a little more daring and want to use whatever is on GitHub, you can have `pip` install right from there:

```
$ pip install git+https://github.com/RIPE-NCC/ripe.atlas.sagan.git
```

### From a Tarball

If for some reason you want to just download the source and install it manually, you can always do that too. Simply `un-tar` the file and run the following in the same directory as `setup.py`:

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

### Troubleshooting

Some setups (like MacOS) have trouble with building the dependencies required for reading SSL certificates. If you don't care about SSL stuff and only want to use `sagan` to say, parse traceroute or DNS results, then you can do the following:

```
$ SAGAN_WITHOUT_SSL=1 pip install ripe.atlas.sagan
```

More information can also be found [here](#).

If you *do* care about SSL and have to use a Mac, then [this issue](#) will likely be of assistance. Essentially, you will need to uninstall Xcode (if it's installed already), then attempt to use `gcc`. This will trigger the OS to ask if you want to install the Xcode compilation tools. Click `install`, and when that's finished, install Sagan with this command:

```
$ CFLAGS="-I/usr/include" pip install ripe.atlas.sagan
```

## Use & Examples

The library contains a full test suite for each measurement type, so if you're looking for examples, it's a good idea to start there. For this document we'll cover basic usage and some simple examples to get you started.

### How To Use This Library

Sagan's sole purpose is to make RIPE Atlas measurements manageable from within Python. You shouldn't have to be fiddling with JSON, or trying to find values that changed locations between firmware versions. Instead, you should always be able to pass in the JSON string and immediately get usable Python objects.



## Important Note

The one thing that tends to confuse people when first trying out Sagan is that this library operates on **single measurement results**, and not a list of results. If you have a list of results (for example, the output of the measurement results API), then you must loop over those results and pass each result to Sagan for parsing.

## Basics

To that end, the interface is pretty simple. If you have a ping measurement result, then use the `PingResult` class to make use of the data:

```
from ripe.atlas.sagan import PingResult

my_result = PingResult('this is where your big JSON blob goes')

my_result.af
# Returns 6

my_result.rtt_median
# Returns 123.456
```

Note that `rtt_median` isn't actually in the JSON data passed in. It's calculated during the parsing phase so you don't need to fiddle with looping over attributes in a list and doing the math yourself.

## Plain Text Not Required

It should be noted that while all of the examples here use a plain text string for our results, Sagan doesn't force you to pass in a string. It's just as happy with a Python dict, the result of already running your result string through `json.loads()`:

```
import json
from ripe.atlas.sagan import PingResult

my_result_dict = json.loads('this is where your big JSON blob goes')
my_result = PingResult(my_result_dict)

my_result.af
# Returns 6

my_result.rtt_median
# Returns 123.456
```

## Agnostic Parsing

There may be a case where you have code that's just expected to parse a result string, without knowing ahead of time what type of result it is. For this we make use of the parent `Result` class' `get()` method:

```
from ripe.atlas.sagan import Result

my_result = Result.get('this is where your big JSON blob goes')

my_result.af
# Returns 6
```

```
my_result.rtt_median
# Returns 123.456
```

As you can see it works just like `PingResult`, but doesn't force you to know its type up front. Note that this does incur a small performance penalty however.

### Errors & Malformations

RIPE Atlas, like the Internet is never 100% what you'd expect. Sometimes your measurement will return an error such as a timeout or DNS lookup problem, and sometimes the data in a result might even be malformed on account of data corruption, damaged probe storage, etc.

And like the most applications on the Internet, Sagan attempts to handle these inconsistencies gracefully. You can decide just how gracefully however.

Say for example you've got a result that looks alright, but the `abuf` value is damaged in some way rendering it unreadable. You'll find that while the `DnsResult` object will not have a `is_malformed=False`, the portion that is unreadable will be set to `True`:

```
from ripe.atlas.sagan import DnsResult
my_result = DnsResult('your JSON blob')

my_result.is_error                # False
my_result.is_malformed            # False
my_result.responses[0].abuf.is_malformed # True
my_result.responses[1].abuf.is_malformed # False
```

You can control what you'd like Sagan to do in these cases by setting `on_malformation=` when parsing:

```
from ripe.atlas.sagan import DnsResult

# Sets is_malformed=True and issues a warning
my_result = DnsResult('your JSON blob')

# Sets is_malformed=True
my_result = DnsResult('your JSON blob', on_malformation=DnsResult.ACTION_IGNORE)

# Sets explodes with a ResultParseError
my_result = DnsResult('your JSON blob', on_malformation=DnsResult.ACTION_FAIL)
```

Similarly, you can do the same thing with `on_error=`, which perform the same way when Sagan encounters an error like a timeout or DNS lookup problem.

Error handling is not yet complete in Sagan, so if you run across a case where it behaves in a way other than what you'd expect, please send a copy of the problematic result to [atlas@ripe.net](mailto:atlas@ripe.net) and we'll use it to update this library.

## Examples

### Parsing Results out of a Local File

Assume for a moment that you've downloaded a bunch of results into a local file using our *fragmented JSON* format. That is, you have in your possession a file that has a separate JSON result on every line. For the purposes of our example we'll call it `file.txt`:

```

from ripe.atlas.sagan import Result

my_results_file = "/path/to/file.txt"
with open(my_results_file) as results:
    for result in results.readlines():
        parsed_result = Result.get(result)
        print(parsed_result.origin)

```

Basically you use Python to open the file (using `with`) and then loop over each line in the file (`.readlines()`), sending each line into Sagan which returns a `parsed_result`. With that result, you can then pull out any of the values you like, using the *Attributes & Methods* documentation as a reference.

## Pulling Directly from the API

A common use case for the parser is to plug it into our RESTful API service. The process for this is pretty simple: fetch a bunch of results, loop over them, and for each one, apply the parser to get the value you want.

Say for example you want to get the checksum value for each result from measurement #1012449. To do this, we'll fetch the latest results from each probe via the `measurement-latest` API, and parse each one to get the checksum values:

```

import requests
from ripe.atlas.sagan import SslResult

source = "https://atlas.ripe.net/api/v1/measurement-latest/1012449/"
response = requests.get(source).json

for probe_id, result in response.items():

    result = result[0]           # There's only one result for each probe
    parsed_result = SslResult(result) # Parsing magic!

    # Each SslResult has n certificates
    for certificate in parsed_result.certificates:
        print(certificate.checksum) # Print the checksum for this certificate

    # Make use of the handy get_checksum_chain() to render the checksum of each_
    ↪ certificate into one string if you want
    print(parsed_result.get_checksum_chain())

```

## Samples from Each Type

### Ping

For more information regarding all properties available, you should consult the *Ping* section of this documentation.:

```

ping_result.packets_sent # Int
ping_result.rtt_median  # Float, rounded to 3 decimal places
ping_result.rtt_average # Float, rounded to 3 decimal places

```

### Traceroute

For more information regarding all properties available, you should consult the *Traceroute* section of this documentation.:

```
traceroute_result.af                # 4 or 6
traceroute_result.total_hops        # Int
traceroute_result.destination_address # An IP address string
```

### DNS

For more information regarding all properties available, you should consult the *DNS* section of this documentation.:

```
dns_result.responses                # A list of Response objects
dns_result.responses[0].response_time # Float, rounded to 3 decimal places
dns_result.responses[0].headers      # A list of Header objects
dns_result.responses[0].headers[0].nscount # The NSCOUNT value for the first header
dns_result.responses[0].questions    # A list of Question objects
dns_result.responses[0].questions[0].type # The TYPE value for the first question
dns_result.responses[0].abuf         # The raw, unparsed abuf string
```

### SSL Certificates

For more information regarding all properties available, you should consult the *SSL Certificate* section of this documentation.:

```
ssl_result.af                # 4 or 6
ssl_result.certificates       # A list of Certificate objects
ssl_result.certificates[0].checksum # The checksum for the first certificate
```

### HTTP

For more information regarding all properties available, you should consult the *HTTP* section of this documentation.:

```
http_result.af                # 4 or 6
http_result.uri                # A URL string
http_result.responses          # A list of Response objects
http_result.responses[0].body_size # The size of the body of the first response
```

## Attributes & Methods

### Common Attributes

All measurement results have a few common properties.

| Property           | Type     | Explanation   |
|--------------------|----------|---|
| raw_data           | dict     | The entire measurement result, as-is from <code>json.loads()</code> |
| created            | datetime | The time at which this result was initiated                         |
| created_timestamp  | int      | A Unix timestamp value for the <code>created</code> attribute       |
| measurement_id     | int      |   |
| probe_id           | int      |   |
| firmware           | int      | The probe firmware release  |
| origin             | str      | The IP address of the probe   |
| seconds_since_sync | int      | The number of seconds since the probe last synchronised its clock   |
| is_malformed       | bool     | Whether the result (or related portion thereof) is unparseable      |
| is_error           | bool     | Whether or not there were errors in parsing/handling this result    |
| error_message      | str      | If the result is an error, the message string is in here            |

## Ping

The simplest measurement type, ping measurement results contain all of the properties *common to all measurements* as well as the following:

| Property            | Type  | Explanation  |
|---------------------|-------|--|
| af                  | int   | The address family. It's always either a 4 or a 6.                                 |
| duplicates          | int   | The number duplicates found  |
| rtt_average         | float |  |
| rtt_median          | float |  |
| rtt_min             | float |  |
| rtt_max             | float |  |
| packets_sent        | int   |  |
| packets_received    | int   |  |
| packet_size         | int   |  |
| destination_name    | str   | The string initially given as the target. It can be an IP address or a domain name |
| destination_address | str   | An IP address represented as a string  |
| step                | int   | The number of seconds between ping requests (interval)                             |
| packets             | list  | A list of ping <i>Packet</i> objects   |

## Packet

Each ping request sends `n` packets, where `n` is a value specified at measurement creation time. We represent these packets as `Packet` objects.

| Property       | Type  | Explanation  |
|----------------|-------|--|
| rtt            | float |  |
| dup            | bool  | Set to <code>True</code> if this packet is a duplicate |
| ttl            | int   |  |
| source_address | str   | An IP address represented as a string                  |

## Traceroute

Probably the largest result type, traceroute measurement results contain all of the properties *common to all measurements* as well as the following:

| Property                 | Type      | Explanation  |
|--------------------------|-----------|--|
| af                       | int       | The address family. It's always either a 4 or a 6.   |
| destination_name         | str       | The string initially given as the target. It can be an IP address or a domain name   |
| destination_address      | str       | An IP address represented as a string  |
| source_address           | str       | An IP address represented as a string  |
| end_time                 | date-time | The time at which the traceroute finished  |
| end_time_timestamp       | int       | A Unix timestamp for the <code>end_time</code> attribute   |
| paris_id                 | int       |  |
| size                     | int       | The packet size  |
| protocol                 | str       | One of ICMP, TCP, UDP  |
| hops                     | list      | A list of <i>Hop</i> objects. If the <code>parse_all_hops</code> parameter is <code>False</code> , this will only contain the last hop.                          |
| total_hops               | int       | The total number of hops   |
| ip_path                  | list      | A list of dicts containing the IPs at each hop. This is just for convenience as all of these values are accessible via the <i>Hop</i> and <i>Packet</i> objects. |
| last_median_rtt          | float     | The median value of all RTTs from the last successful hop  |
| destination_ip_responded | bool      | Set to <code>True</code> if the last hop was a response from the destination IP  |
| last_hop_responded       | bool      | Set to <code>True</code> if the last hop was a response at all   |
| is_success               | bool      | Set to <code>True</code> if the traceroute finished successfully   |
| last_hop_errors          | list      | A list of last hop's errors  |

It is also possible to supply the following parameter to control parsing of Traceroute results:

| Parameter                   | Type | Default           | Explanation  |
|-----------------------------|------|-------------------|--|
| <code>parse_all_hops</code> | bool | <code>True</code> | Set to <code>False</code> to stop parsing <i>Hop</i> objects after the <code>last_*</code> properties (see above) have been set. This will cause <code>hops</code> to only contain the last <i>Hop</i> . |

## Hop

Each hop in the traceroute is available as a *Hop* object.

| Property   | Type  | Explanation                                |
|------------|-------|--|
| index      | int   | The hop number, starting with 1            |
| packets    | list  | A list of traceroute <i>Packet</i> objects |
| median_rtt | float | The median value of all RTTs of the hop    |

## Packet

| Property                | Type        | Explanation   |
|-------------------------|-------------|---|
| origin                  | str         | The IP address of where the packet is coming from   |
| rtt                     | float       |   |
| size                    | int         |   |
| ttl                     | int         |   |
| arrived_late_by         | int         | If the packet arrived late, this number represents “how many hops ago” this packet was sent |
| internal_ttl            | int         | The time-to-live for the packet that triggered the error ICMP. The default is 1             |
| destination_option_size | int         | The size of the IPv6 destination option header  |
| hop_by_hop_option_size  | int         | The size of the IPv6 hop-by-hop option header   |
| icmp_header             | Icmp-Header | See <i>IcmpHeader</i> below   |

## IcmpHeader

This class is slightly different than other parts of Sagan as in `objects` we find a complex generic list containing generic dictionaries pulled directly from the JSON blob. The decision not to further parse this blob into separate Python models was made based on the assumption that much of this section is very edge-case and the contents are present sporadically.

If however there is a demand for further development of this portion of the result, we can expand it. Until then though, `IcmpHeader` is a very simple class, the majority of data living in `objects`.

For further information about this portion of a traceroute result, you should consult our [data structure documentation](#)

| Property | Type | Explanation  |
|----------|------|--|
| version  | int  | RFC4884 version  |
| rfc4884  | bool | True if length indication is present, False otherwise                              |
| objects  | list | As mentioned above a complete dump of whatever is in the <code>obj</code> property |

## DNS

The most complicated result type, `dns` measurement results contain all of the properties *common to all measurements* as well as the following:

| Property  | Type | Explanation                                       |
|-----------|------|---|
| responses | list | A list of DNS <i>Response</i> objects (see below) |

## Response

Most DNS measurement results consist of a single response, but in some cases, there may be more than one. Regardless, every `Response` instance has the following properties:

| Property            | Type    | Explanation  |
|---------------------|---------|--|
| raw_data            | dict    | The fragment of the initial JSON that pertains to this response  |
| af                  | int     | The address family. It's always either a 4 or a 6.   |
| destination_address | str     | An IP address represented as a string  |
| source_address      | str     | An IP address represented as a string  |
| protocol            | str     | One of TCP, UDP  |
| abuf                | Message | See <i>Message</i> below   |
| qbuf                | Message | See <i>Message</i> below   |
| response_time       | float   | Time, in milliseconds until the response was received  |
| response_id         | int     | The sequence number of this result within a group of results, available if the resolution was done by the probe's local resolver |

## Message

Responses can contain either an `abuf` or a `qbuf` which are both `Message` objects. If you want the string representation, simply cast the object as a string with `str()`.

| Property    | Type   | Explanation   |
|-------------|--------|---|
| raw_data    | dict   | The fragment of the initial JSON that pertains to this response |
| header      | Header | See <i>Header</i> below   |
| edns0       | Edns0  | See <i>EDNS0</i> below, if any                                  |
| questions   | list   | A list of <i>Question</i> objects                               |
| answers     | list   | A list of <i>Answer</i> objects, if any                         |
| authorities | list   | A list of <i>Answer</i> objects, if any                         |
| additional  | list   | A list of <i>Answer</i> objects, if any                         |

## A note on pre-calculated values

By default, when you pass a result into Sagan, it will attempt to parse the `abuf` and `qbuf` strings (if any) into `Message` objects. However, some of the values in that `abuf` may have already been pre-calculated and stored alongside the other attributes in the result. Many `Header` values for example, can be found in the raw result (outside of the `abuf` string), so parsing the `abuf` for these values is redundant and potentially unnecessary if these values are all you need.

For this case, Sagan supports passing `parse_buf=False` to the `DnsResult` class. If you opt for this method, the `abuf` will not be parsed, and any values not immediately available in the result will return `None`. For example:

```
from ripe.atlas.sagan import DnsResult
my_result = DnsResult(
    '<some result data including name, type, and rdata, but not ttl or class>',
    parse_buf=False
)
result.responses[0].abuf.answers[0].name           # "version.bind"
result.responses[0].abuf.answers[0].klass         # None
result.responses[0].abuf.answers[0].rd_length    # None
result.responses[0].abuf.answers[0].type         # "TXT"
result.responses[0].abuf.answers[0].ttl          # None
result.responses[0].abuf.answers[0].data         # "Some RDATA value"
```

Note also that `Result.get()` accepts `parse_buf=` as well:



```

from ripe.atlas.sagan import Result
my_result = Result.get(
    '<some result data including name, type, and rdata, but not ttl or class>',
    parse_buf=False
)
result.responses[0].abuf.answers[0].name # "version.bind"
...

```

## Header

All of these properties conform to [RFC 1035](#), so we won't go into detail about them here.

| Property    | Type | Explanation   |
|-------------|------|---|
| raw_data    | dict | The portion of the parsed abuf that represents this section |
| aa          | bool |   |
| qr          | bool |   |
| nscount     | int  | Otherwise known as the nameserver count or authority count. |
| qdccount    | int  |   |
| ancount     | int  |   |
| tc          | bool |   |
| rd          | bool |   |
| arcount     | int  |   |
| return_code | str  |   |
| opcode      | str  |   |
| ra          | bool |   |
| z           | int  |   |
| id          | int  |   |

## Question

The question section of the response.

**NOTE:** In keeping with Python conventions, we use the `propertyname klass` here instead of the more intuitive (and illegal in Python) `class`. It may be confusing for non-Python programmers, but unfortunately it's a limitation of the language.

| Property | Type | Explanation   |
|----------|------|---|
| raw_data | dict | The portion of the parsed abuf that represents this section |
| klass    | str  | The CLASS value, spelt this way to conform to Python norms  |
| type     | str  |   |
| name     | str  |   |

## Answer

The answer section of the response.

**NOTE:** In keeping with Python conventions, we use the `propertyname klass` here instead of the more intuitive (and illegal in Python) `class`. It may be confusing for non-Python programmers, but unfortunately it's a limitation of the language.

| Property  | Type | Explanation   |
|-----------|------|---|
| raw_data  | dict | The portion of the parsed abuf that represents this section |
| klass     | str  | The CLASS value, spelt this way to conform to Python norms  |
| type      | str  |   |
| name      | str  |   |
| ttl       | int  |   |
| address   | str  | An IP address   |
| rd_length | int  |   |

There is a different sub-class of `Answer` for every DNS answer type. These are all briefly outlined below.

### AAnswer & AAAAAnswer

Both of these classes have only one additional property to their parent `Answer` class: `address`.

| Property | Type | Explanation          |
|----------|------|----------------------|
| answer   | str  | The address response |

### NsAnswer & CnameAnswer

Both of these subclasses only have one additional property: `target`.

| Property | Type | Explanation               |
|----------|------|---------------------------|
| target   | str  | The address of the target |

### MxAnswer

| Property       | Type | Explanation           |
|----------------|------|-----------------------|
| preference     | int  | The preference number |
| mail_exchanger | str  | The exchanger name    |

### SoaAnswer

There are a lot of additional properties for SOA answers, as well as a few aliases for people who like human-readable names.

| Property           | Type | Explanation                       |
|--------------------|------|-----------------------------------|
| mname              | str  | The master server name            |
| rname              | str  | The maintainer name               |
| serial             | int  |                                   |
| refresh            | int  |                                   |
| retry              | int  |                                   |
| expire             | int  |                                   |
| minimum            | int  | The negative TTL                  |
| master_server_name | str  | An alias for <code>mname</code>   |
| maintainer_name    | str  | An alias for <code>rname</code>   |
| negative_ttl       | str  | An alias for <code>minimum</code> |
| nxdomain           | str  | An alias for <code>minimum</code> |

## DsAnswer

| Property       | Type |
|----------------|------|
| tag            | int  |
| algorithm      | int  |
| digest_type    | int  |
| delegation_key | str  |

## DnskeyAnswer

| Property  | Type |
|-----------|------|
| flags     | int  |
| algorithm | int  |
| protocol  | int  |
| key       | str  |

## TxtAnswer

A class for DNS TXT responses, `TxtAnswer` has all of the properties of an `Answer` class, but with two additional properties:

| Property    | Type | Explanation   |
|-------------|------|---|
| data        | list | The response text, represented as a list of strings, though in most cases, the list has only one element. |
| data_string | str  | The string representation of <code>data</code> , joining all elements of the list with a space.           |

## RRSigAnswer

| Property             | Type |
|----------------------|------|
| type_covered         | str  |
| algorithm            | int  |
| labels               | int  |
| original_ttl         | int  |
| signature_expiration | int  |
| signature_inception  | int  |
| key_tag              | int  |
| signer_name          | str  |
| signature            | str  |

Note that `RRSigAnswer`'s have a special string representation, where the values of `type_covered`, `algorithm`, `labels`, `original_ttl`, `signature_expiration`, `signature_inception`, `key_tag`, `signer_name`, and `signature` are all concatenated with spaces.

## NsecAnswer

| Property         | Type |
|------------------|------|
| next_domain_name | str  |
| types            | list |

### Nsec3Answer

| Property       | Type |
|----------------|------|
| hash_algorithm | int  |
| flags          | int  |
| iterations     | int  |
| salt           | str  |
| hash           | str  |
| types          | list |

### Nsec3ParamAnswer

| Property   | Type |
|------------|------|
| algorithm  | int  |
| flags      | int  |
| iterations | int  |
| salt       | str  |

### PtrAnswer

| Property | Type |
|----------|------|
| target   | str  |

### SrvAnswer

| Property | Type |
|----------|------|
| priority | int  |
| weight   | int  |
| port     | int  |
| target   | str  |

### SshfpAnswer

| Property    | Type |
|-------------|------|
| algorithm   | int  |
| digest_type | int  |
| fingerprint | str  |

### TlsaAnswer

| Property                    | Type |
|-----------------------------|------|
| certificate_usage           | int  |
| selector                    | int  |
| matching_type               | int  |
| certificate_associated_data | str  |

## HinfoAnswer

| Property | Type |
|----------|------|
| cpu      | str  |
| os       | str  |

## EDNS0

The optional EDNS0 section of the response.

| Property             | Type | Explanation   |
|----------------------|------|---|
| raw_data             | dict | The portion of the parsed abuf that represents this section |
| extended_return_code | int  |   |
| name                 | str  |   |
| type                 | str  |   |
| udp_size             | int  |   |
| version              | int  |   |
| z                    | int  |   |
| options              | list | A list of <i>Option</i> objects                             |

## Option

| Property | Type | Explanation  |
|----------|------|--|
| raw_data | dict | The portion of the EDNS0 section that represents this option |
| nsid     | str  |  |
| code     | int  |  |
| length   | int  |  |
| name     | str  |  |

## SSL Certificate

SSL certificate measurement results contain all of the properties *common to all measurements* as well as the following:

| Property            | Type  | Explanation   |
|---------------------|-------|---|
| af                  | int   | The address family. It's always either a 4 or a 6.  |
| destination_name    | str   | The string initially given as the target. It can be an IP address or a domain name  |
| destination_address | str   | An IP address   |
| source_address      | str   | An IP address   |
| port                | int   | The port number   |
| method              | str   | This should always be "SSL"   |
| version             | str   |   |
| response_time       | float | Time, in milliseconds until the response was received   |
| time_to_connect     | float | Time, in milliseconds until the connection was established  |
| certificates        | list  | A list of <i>Certificate</i> objects  |
| is_signed           | bool  | Set to <code>True</code> if the certificate is self-signed  |
| checksum_chain      | str   | A list of all checksums for all certificates in this result, joined with the arbitrary string <code>::</code> . This can come in handy when you're trying to compare checksums of multiple results. |

## Certificate

Each SSL certificate measurement result can contain multiple `Certificate` objects.

| Property                      | Type      | Explanation  |
|-------------------------------|-----------|--|
| <code>raw_data</code>         | dict      | The fragment of the initial JSON that pertains to this response  |
| <code>subject_cn</code>       | str       | The subject's common name  |
| <code>subject_o</code>        | str       | The subject's organisation   |
| <code>subject_c</code>        | str       | The subject's country  |
| <code>issuer_cn</code>        | str       | The issuer's common name   |
| <code>issuer_o</code>         | str       | The issuer's organisation  |
| <code>issuer_c</code>         | str       | The issuer's country   |
| <code>valid_from</code>       | date-time |  |
| <code>valid_until</code>      | date-time |  |
| <code>check-sum_md5</code>    | str       | The md5 checksum   |
| <code>check-sum_sha1</code>   | str       | The sha1 checksum  |
| <code>check-sum_sha256</code> | str       | The sha256 checksum  |
| <code>has_expired</code>      | bool      | Set to <code>True</code> if the certificate is no longer valid   |
| <code>extensions</code>       | dict      | Parsed extensions. For now it can only be <code>subjectAltName</code> , which is a list of names contained in the SAN extension, if that exists. |

## HTTP

HTTP measurement results contain all of the properties *common to all measurements* as well as the following:

| Property               | Type | Explanation                       |
|------------------------|------|-----------------------------------|
| <code>uri</code>       | str  |                                   |
| <code>method</code>    | str  | The HTTP method                   |
| <code>responses</code> | list | A list of <i>Response</i> objects |

## Response

Each HTTP measurement result can contain multiple `Response` objects.

| Property                         | Type  | Explanation  |
|----------------------------------|-------|--|
| <code>raw_data</code>            | dict  | The portion of the JSON that pertains to this response |
| <code>af</code>                  | int   | The address family. It's always either a 4 or a 6.     |
| <code>body_size</code>           | int   | The total number of bytes in the body                  |
| <code>head_size</code>           | int   | The total number of bytes in the head                  |
| <code>destination_address</code> | str   | An IP address  |
| <code>source_address</code>      | str   | An IP address  |
| <code>code</code>                | int   | The HTTP response code                                 |
| <code>response_time</code>       | float | Time, in milliseconds until the response was received  |
| <code>version</code>             | str   | The HTTP version                                       |

## NTP

NTP measurement results contain all of the properties *common to all measurements* as well as the following:

| Property              | Type  | Explanation  |
|-----------------------|-------|--|
| leap_second_indicator | str   | Leap second indicator  |
| poll                  | int   | Poll interval  |
| precision             | float |  |
| protocol              | str   | UDP  |
| reference_id          | str   | Reference id returned by server                                  |
| reference_time        | float | The NTP time the server last contacted the reference time source |
| root_delay            | float | Round trip time from the server to the reference time source     |
| root_dispersion       | float | Accuracy of server's clock                                       |
| stratum               | int   | How far in hops is server from reference time source             |
| version               | int   | The NTP version  |
| mode                  | str   | Ntp communication mode. Usually <code>server</code>              |
| rtt_median            | float | The median value of packets' rtt                                 |
| offset_median         | float | The median value of the packets' offset                          |
| packets               | list  | A list of ntp <a href="#">Response</a> objects                   |

## Response

Each HTTP measurement result can contain multiple `Response` objects.

| Property                   | Type          | Explanation  |
|----------------------------|---------------|--|
| raw_data                   | dict          | The portion of the JSON that pertains to this response   |
| offset                     | float         | The NTP offset   |
| rtt                        | float         | The response time  |
| final_timestamp            | float         | A full-precision Unix timestamp for when the NTP client received the response                  |
| origin_timestamp           | float         | A full-precision Unix timestamp for when the NTP client send packet to the server              |
| re-ceived_timestamp        | float         | A full-precision Unix timestamp for when the NTP server received the request                   |
| transmit-<br>ted_timestamp | float         | A full-precision Unix timestamp for when the NTP server transmitted the response               |
| final_time                 | date-<br>time | A Python datetime object with limited precision[1] based on <code>final_timestamp</code>       |
| origin_time                | date-<br>time | A Python datetime object with limited precision[1] based on <code>origin_timestamp</code>      |
| received_time              | date-<br>time | A Python datetime object with limited precision[1] based on <code>received_timestamp</code>    |
| transmitted_time           | date-<br>time | A Python datetime object with limited precision[1] based on <code>transmitted_timestamp</code> |

## How To Contribute

We would love to have contributions from everyone and no contribution is too small. Please submit as many fixes for typos and grammar bloopers as you can!

To make participation in this project as pleasant as possible for everyone, we adhere to the [Code of Conduct](#) by the Python Software Foundation.

The following steps will help you get started:

Fork, then clone the repo:

```
$ git clone git@github.com:your-username/ripe.atlas.sagan.git
```

Make sure the tests pass beforehand:

```
$ tox
```

or

```
$ nosetests tests/
```

Make your changes. Include tests for your change. Make the tests pass:

```
$ tox
```

or

```
$ nosetests tests/
```

Push to your fork and [submit a pull request](#).

Here are a few guidelines that will increase the chances of a quick merge of your pull request:

- *Always* try to add tests and docs for your code. If a feature is tested and documented, it's easier for us to merge it.
- Follow [PEP 8](#).
- Write [good commit messages](#).
- If you change something that is noteworthy, don't forget to add an entry to the [changes](#).

---

### Note:

- If you think you have a great contribution but aren't sure whether it adheres – or even can adhere – to the rules: **please submit a pull request anyway!** In the best case, we can transform it into something usable, in the worst case the pull request gets politely closed. There's absolutely nothing to fear.
  - If you have a great idea but you don't know how or don't have the time to implement it, please consider opening an issue and someone will pick it up as soon as possible.
- 

Thank you for considering a contribution to this project! If you have any questions or concerns, feel free to reach out the RIPE Atlas team via the [mailing list](#), [GitHub Issue Queue](#), or [messenger pigeon](#) – if you must.

## Changelog

- **1.2.1**
  - Add support for non-DNS names in subjectAltName extensions
- **1.2**
  - Replaced pyOpenSSL with cryptography
  - Added parsing of subjectAltName X509 extension
- **1.1.11**



- Added first version of WiFi results
- **1.1.10**
  - Added a *parse\_all\_hops* kwarg to the Traceroute class to tell Sagan to stop parsing Hops and Packets once we have all of the last hop statistics (default=True)
  - Remove dependency on IPy: we were using it for IPv6 canonicalization, but all IPv6 addresses in results should be in canonical form to start with.
- **1.1.9**
  - Removed the *parse\_abuf* script because no one was using it and its Python3 support was suspect anyway.
- **1.1.8**
  - Handle case where a traceroute result might not have *dst\_addr* field.
- **1.1.7**
  - Change condition of traceroute’s *last\_hop\_responded* flag.
  - Add couple of more traceroute’s properties. *is\_success* and *last\_hop\_errors*.
  - Add tests to the package itself.
- **1.1.6**
  - Fix for [Issue #56](#) a case where the *qbuf* value wasn’t being properly captured.
  - Fixed small bug that didn’t accurately capture the *DO* property from the *qbuf*.
- **1.1.5**
  - We now ignore so-called “late” packets in traceroute results. This will likely be amended later as future probe firmwares are expected to make better use of this value, but until then, Sagan will treat these packets as invalid.
- **1.1.4**
  - Added a *type* attribute to all *Result* subclasses
  - Added support for a lot of new DNS answer types, including *NSEC*, *PTR*, *SRV*, and more. These answers do not yet have a complete string representation however.
- **1.1.3**
  - Changed the name of *TracerouteResult.rtt\_median* to *TracerouteResult.last\_rtt\_median*.
  - Modified the *DnsResult* class to allow the “bubbling up” of error statuses.
- **1.1.2**
  - We skipped this number for some reason :-/
- **1.1.1**
  - Fixed a [string representation bug](#) found by [iortiz](#)
- **1.1.0**
  - **Breaking Change:** the *Authority* and *Additional* classes were removed, replaced with the appropriate answer types. For the most part, this change should be invisible, as the common properties are the same, but if you were testing code against these class types, you should consider this a breaking change.

- **Breaking Change:** The `__str__` format for `DNS RrsigAnswer` to conform the output of a typical `dig` binary.
- Added `__str__` definitions to DNS answer classes for use with the toolkit.
- In an effort to make Sagan (along with Cousteau and the toolkit) more portable, we dropped the requirement for the `arrow` package.
- **1.0.0**
  - 1.0! w00t!
  - **Breaking Change:** the `data` property of the `TxtAnswer` class was changed from a string to a list of strings. This is a correction from our own past deviation from the RFC, so we thought it best to conform as part of the move to 1.0.0
  - Fixed a bug where non-ascii characters in DNS TXT answers resulted in an exception.
- **0.8.2**
  - Fixed a bug related to non-ascii characters in SSL certificate data.
  - Added a wrapper for json loaders to handle differences between `ujson` and the default `json` module.
- **0.8.1**
  - Minor fix to make all `Result` objects properly JSON serialisable.
- **0.8.0**
  - Added `iertiz`'s patch for `flags` and `flags` and `sections` properties on `DNS Answer` objects.
- **0.7.1**
  - Changed `README.md` to `README.rst` to play nice with `pypi`.
- **0.7**
  - Added `pierky`'s new `RRSigAnswer` class to the `dns` parser.
- **0.6.3**
  - Fixed a bug in how Sagan deals with inappropriate firmware versions
- **0.6.2**
  - Added `pierky`'s fix to fix AD and CD flags parsing in DNS Header
- **0.6.1**
  - Added `rtt_min`, `rtt_max`, `offset_min`, and `offset_max` to `NTPResult`
- **0.6.0**
  - Support for NTP measurements
  - Fixes for how we calculate median values
  - Smarter `setup.py`
- **0.5.0**
  - Complete Python3 support!
- **0.4.0**
  - Added better Python3 support. Tests all pass now for ping, traceroute, ssl, and http measurements.
  - Modified `traceroute` results to make use of `destination_ip_responded` and `last_hop_responded`, deprecating `target_responded`. See the docs for details.

- **0.3.0**
  - Added support for making use of some of the pre-calculated values in DNS measurements so you don't have to parse the abuf if you don't need it.
  - Fixed a bug in the abuf parser where a variable was being referenced by never defined.
  - Cleaned up some of the abuf parser to better conform to pep8.
- **0.2.8**
  - Fixed a bug where DNS TXT results with class IN were missing a `.data` value.
  - Fixed a problem in the SSL unit tests where `\n` was being misinterpreted.
- **0.2.7**
  - Made abuf more robust in dealing with truncation.
- **0.2.6**
  - Replaced `SslResult.get_checksum_chain()` with the `SslResult.checksum_chain` property.
  - Added support for catching results with an `err` property as an actual error.
- **0.2.5**
  - Fixed a bug in how the `on_error` and `on_malformation` preferences weren't being passed down into the subcomponents of the results.
- **0.2.4**
  - Support for `seconds_since_sync` across all measurement types
- **0.2.3**
  - “Treat a missing Type value in a DNS result as a malformation” (Issue #36)
- **0.2.2**
  - Minor bugfixes
- **0.2.1**
  - Added a `median_rtt` value to traceroute Hop objects.
  - Smarter and more consistent error handling in traceroute and HTTP results.
  - Added an `error_message` property to all objects that is set to `None` by default.
- **0.2.0**
  - Totally reworked error and malformation handling. We now differentiate between a result (or portion thereof) being malformed (and therefore unparsable) and simply containing an error such as a timeout. Look for an `is_error` property or an `is_malformed` property on every object to check for it, or simply pass `on_malformation=Result.ACTION_FAIL` if you'd prefer things to explode with an exception. See the documentation for more details
  - Added lazy-loading features for parsing abuf and qbuf values out of DNS results.
  - Removed the deprecated properties from `dns.Response`. You must now access values like `edns0` from `dns.Response.abuf.edns0`.
  - More edge cases have been found and accommodated.
- **0.1.15**
  - Added a bunch of abuf parsing features from [b4ldr](#) with some help from [phicoh](#).

- **0.1.14**
  - Fixed the deprecation warnings in `DnsResult` to point to the right place.
- **0.1.13**
  - Better handling of `DNSResult` errors
  - Rearranged the way `abufs` were handled in the `DnsResult` class to make way for `qbuf` values as well. The old method of accessing `header`, `answers`, `questions`, etc is still available via `Response`, but this will go away when we move to 0.2. Deprecation warnings are in place.
- **0.1.12**
  - Smarter code for checking whether the target was reached in `TracerouteResults`.
  - We now handle the `destination_option_size` and `hop_by_hop_option_size` values in `TracerouteResult`.
  - Extended support for ICMP header info in `traceroute Hop` class by introducing a new `IcmpHeader` class.
- **0.1.8**
  - Broader support for SSL checksums. We now make use of `md5` and `sha1`, as well as the original `sha256`.