Zeroguard Recon API

Release 0.0.1-dev.1

Jun 11, 2021

Introduction

ZeroGuard Recon is a threat intelligence platform that facilitiates data discovery, analysis and attribution using serveral sources such as DNS, certificate transparency logs, WHOIS, IP reputation and much more.

This is the main page of ZeroGuard Recon public API documentation which is aimed at providing a detailed and structured reference for all API users. Below is a table of contents that should help you to get started in no time.

Warning: The documentation is still very much work in progress. Please do not attempt to implement any API clients until this warning disappears. Current API structure is by no means set in stone and may change without a further notice.

Note: If you are interested in this project do not hesitate to get in touch by sending an email to info@zeroguard.com or filling out a form on our website (scroll to the bottom of a page).

CHAPTER 1

Features Overview

Todo: Fill out a features overview page (this one)

CHAPTER 2

Versioning

ZeroGuard Recon is an actively evolving platform that thrives to deliver as many useful features to our customers as it is physically possible to implement in a give time frame. As such we cannot commit to maintaining backwards compatibility for as long as there are any major features we would like to add to our platform.

That's being said, the burden of maintaining all API clients and making sure they are fully functional and are compatible with the latest features are completely on our shoulders.

A public roadmap of features that are planned for a release will be available in the near future.

Warning: As it is mentioned above, this API is in an early access and may change without further notice at any given moment of time. We apologise for any caused inconvenience and hope for your understanding.

CHAPTER 3

Authentication

Todo: Fill out an authentication documentation page

CHAPTER 4

Data Types

Data types represent all different entities that can be referenced (see *Object References* for details) in the API response object. Each data type defines a unique value for a `type` field as well as any other type-specific fields.

4.1 IPv4 Address

`IPv4 Address`
IPv4 address.

Object Properties

- **`type`** (*`string`*) – Type definition. Always present and has a value of `ipv4`.
- **`address`** (*`string`*) – IPv4 address. Always present.
- **`closest_prefix`** (*`Prefix Reference`*) – A reference to the smallest network prefix which contains this IP address. Always present.
- **`prefixes`** (*list of `Prefix Reference`*) – A list of references to multiple network prefixes which together represent a full prefix chain for this IP (from the smallest to 0.0.0.0/0). Always present.
- **`reputation`** (*list of `IP Reputation Entry`*) – List of IP reputation entries containing information from reputation feeds in which this IP address was observed throughout its lifespan. Always present.

`Prefix Reference`
A reference to *Network Prefix* which contains an IP address.

Object Properties

- **`_ref`** (*`int`*) – Reference number. Always present.

`IP Reputation Entry`
IP reputation feed entry. Instances of this type are always located in `reputation` list of *`IPv4 Address`* or *`IPv6 Address`* objects as they are generated dynamically for a specific IP returned in the API response.

Object Properties

- **name** (*string*) – Name of IP reputation feed to which this entry belongs to. Always pressent.
- **current** (*bool*) – Flag which determines whether this reputation feed entry is current (exists in a source reputation feed when the response is returned). Always present.
- **first_seen** (*int*) – Unix timestamp of when this entry was first observed. Always present.
- **last_seen** (*int*) – Unix timestamp of when this entry was last observed. Always present.

Todo: Replace IPv4 address data type example with an actual output from the API. Currently it is a stub.

Example of *IPv4 Address* data type instance (truncated for clarity):

```
{
    "type": "ipv4",
    "address": "8.8.8.8",
    "closest_prefix": {"_ref": 1},
    "prefixes": [
        {"_ref": 1},
        {"_ref": 3},
        {"_ref": 4}
    ],
    "reputation": [
        {
            "name": "firehol-coinbl-hosts",
            "current": false,
            "first_seen": 1584712048,
            "last_seen": 1584720037

        },
        {
            "name": "firehol-dshield-top-1000",
            "current": true,
            "first_seen": 1584714021,
            "last_seen": 1584720037
        }
    ]
}
```

4.2 IPv6 Address

IPv6 address. Structure is near identical to *IPv4 Address* thus a lot of nested object definitions are re-used.

IPv6 Address
IPv6 address.

Object Properties

- **type** (*string*) – Type definition. Always present and has a value of `ipv6`.
- **address** (*string*) – IPv6 address. Always present.

- **closest_prefix** (*Prefix Reference*) – A reference to the smallest network prefix which contains this IP address. Always present.
- **prefixes** (*list of Prefix Reference*) – A list of references to multiple network prefixes which together represent a full prefix chain for this IP (from the smallest to 0.0.0.0/0). Always present.
- **reputation** (*list of IP Reputation Entry*) – List of IP reputation entries containing information from reputation feeds in which this IP address was observed throughout its lifespan. Always present.

Todo: Replace IPv6 address data type example with an actual output from the API. Currently it is a stub.

Example (truncated for clarity):

```
{
    "type": "ipv6",
    "address": "2001:4860:4860::8888",
    "closest_prefix": {"_ref": 3},
    "prefixes": [
        {"_ref": 3},
        {"_ref": 4},
        {"_ref": 12}
    ],
    "reputation": [
        {
            "name": "firehol-coinbl-hosts",
            "current": false,
            "first_seen": 1584712048,
            "last_seen": 1584720037

        },
        {
            "name": "firehol-dshield-top-1000",
            "current": true,
            "first_seen": 1584714021,
            "last_seen": 1584720037
        }
    ]
}
```

4.3 Network Prefix

Todo: Replace Network Prefix data type stub with an actual definition

Network Prefix
Network prefix.

Object Properties

- **type** (*string*) – Type definition. Always present and has a value of `netpref`.
- **prefix** (*string*) – Actual value of a network prefix.

Example:

```
{
    "type": "netpref",
    "prefix": "0.0.0.0/0"
}
```

4.4 Subdomain

Subdomain
Internet subdomain.

Object Properties

- **type** (*string*) – Type definition. Always present and has a value of `subdomain`.
- **name** (*string*) – Name of a subdomain. Always present.

Example:

```
{
    "type": "subdomain",
    "name": "foo.example.com"
}
```

CHAPTER 5

Errors

All error responses sent by the API have a unified structure. Each error response contains two contextual keys: `query` and `error.error_context`. The former just mirrors specified request parameters while the latter contains contextual error data that is specific for an endpoint to which the request was sent.

See *Error Types* for a list of all defined errors.

5.1 Structure

`Error Response`
General structure of an error response.

Object Properties

- **`query`** (*`object`*) – Contextual information about the query that was performed. Essentially, this object just mirrors request parameters that were specified (after a disambiguation and resolution process). Object structure is defined individually for each API endpoint. See Query Object for a half-decent generalization of this object's structure. May be absent.
- **`error`** (*`Error Object`*) – Error object that was returned. Always present.

`Error Object`
Error object structure.

Object Properties

- **`error_name`** (*`string`*) – Name of an error that occured. Always present.
- **`error_desc`** (*`string`*) – Description of an error that occured. It is static and is defined by the error itself, not a request that was sent. Always present.
- **`error_context`** (*`object`*) – A nested object with a contextual information about the error. Object structure is defined individually for each API endpoint. May be absent.

5.2 Examples

An example of a full-fledged error response:

```
{
    "query": {
        "action": "search_subdomains",
        "parameters": {
            "domain": "example.com",
            "iv.history": true
        }
    },
    "error": {
        "error_name": "bad_request",
        "error_description": "Request is malformed",
        "error_context": {
            "invalid_parameter_name": "iv.history",
            "valid_parameter_names": [
                "ipv4.history",
                "ipv4.latest",
                "ipv4.live",
                "ipv4.meta",
                "ipv4.oldest",
                "ipv6.history",
                "ipv6.latest",
                "ipv6.live",
                "ipv6.meta",
                "ipv6.oldest"
            ]
        }
    }
}
```

5.3 Error Types

The majority of error types are tied to a specific HTTP status code though this is not always the case (i.e. both *Empty Result* and *No Such Endpoint* are returned with 404 Not Found). It is advised to always check the response body to determine which type of error has occured.

While the majority of API errors are documented here, there is a chance that client will receive an undocumented error. These can be treated according to HTTP response status they are returned with completely ignoring the response body.

5.3.1 Bad Request

Request object structure is malformed. This error usually contains more information about what is exactly malformed in *error.error_context* object. Context object structure usually differs depending on an endpoint.

HTTP Status Code	Error Name	Error Description
400 Bad Request	`bad_request`	`Request object structure is malformed`

Example:

```
{
    "query": {
        "action": "search_subdomains",
        "parameters": {
            "domain": "69"
        }
    },
    "error": {
        "error_name": "bad_request",
        "error_description": "Request is malformed",
        "error_context": {
            "invalid_parameter_value": "69"
        }
    }
}
```


5.3.2 Empty Result

Request was successfully processed but yielded no results. This error is often sent in conjunction with 404 Not Found HTTP status code thus making it similar to *No Such Endpoint*. Do check the response body to correctly handle these errors.

HTTP Status Code	Error Name	Error Description
404 Not Found	`empty_result`	`Request produced no results`

Example:


```
{
    "query": {
        "action": "search_subdomains",
        "parameters": {
            "domain": "abcabcabcabcabcabcabcabcabcabcabcabcabc.io",
            "ipv4.history": false,
            "ipv4.latest": true,
            "ipv4.live": false,
            "ipv4.meta": false,
            "ipv4.oldest": true,
            "ipv6.history": false,
            "ipv6.latest": true,
            "ipv6.live": false,
            "ipv6.meta": false,
            "ipv6.oldest": true
        }
    },
    "error": {
        "error_name": "empty_result",
        "error_description": "Request produced no results"
    }
}
```


5.3.3 Internal Server Error

Internal server error. This is most probably a bug. See *Bugs Reporting* for more information about how to report bugs and security vulnerabilities.

HTTP Status Code	Error Name	Error Description
500 Internal Server Error	`internal_server_error`	`API server failed to process the request`

Example:

```
{
    "error": {
        "error_name": "internal_server_error",
        "error_description": "API server failed to process the request"
    }
}
```

5.3.4 Method Not Allowed

Requested endpoint does not support HTTP method that was used. Query Object is not returned when this error occurs.

HTTP Status Code	Error Name	Error Description
405 Method Not Allowed	`method_not_allowed`	`Request method is not supported by this endpoint`

Example:

```
{
    "error": {
        "error_name": "method_not_allowed",
        "error_description": "Request method is not supported by this endpoint"
    }
}
```

5.3.5 No Such Endpoint

Requested API endpoint does not exist. Query Object is not returned when this error occurs.

HTTP Status Code	Error Name	Error Description
404 Not Found	`no_such_endpoint`	`Requested API endpoint does not exist`

Example:

```
{
    "error": {
        "error_name": "no_such_endpoint",
        "error_description": "Requested API endpoint does not exist"
    }
}
```

5.3.6 Rate Limit Exceeded

HTTP Status Code	Error Name	Error Description
429 Too Many Requests	`rate_limit_exceeded`	`API rate limit was exceeded`

Todo: Propertly document Rate Limit Exceeded error type

5.3.7 Processing Timeout

Note: This error indicates that a hard limit for resources utilization was reached and no further processing will be made for this request. This is a current limitation of the API. Processing timeout will be increasing in the near future.

Request took too long to process and was abandoned by the API server. No results will be returned.

HTTP Status Code	Error Name	Error Description
501 Not Implemented	`processing_timeout`	`Request processing took too long`


```
{
    "query": {
        "action": "search_subdomains",
        "parameters": {
            "domain": "example.com",
            "ipv4.history": true,
            "ipv4.latest": true,
            "ipv4.live": true,
            "ipv4.meta": true,
            "ipv6.history": true,
            "ipv4.oldest": true,
            "ipv6.latest": true,
            "ipv6.live": true,
            "ipv6.meta": true,
            "ipv6.oldest": true
        }
    },
    "error": {
        "error_name": "processing_timeout",
        "error_description": "Request processing took too long"
    }
}
```

CHAPTER 6

Rate Limits

Todo: Fill out rate limits documentation

CHAPTER 7

Object References

Todo: Object references explanation sucks. Make it better.

Due to a highly nested and interconnected nature of the data provided by the API, there is a need to avoid data duplication and deep nesting in a response object structure. API utilizes object references to achieve that.

It is said that JSON object references another object if and only if it contains `_ref` field.

The easiest way to understand how references work is to explain how they are generated:

1. Retrieve all matching data for a given request from a storage cluster
2. Assign a unique natural number as an ID for each data object in the results
3. Replace all references inside data objects with a newly generated reference numbers that are specific only to this response object.

The result is a flat response object structure which allows each data object to reference multiple other data objects without any data duplication.

Note: Reference objects never contain values that are specific to the request and its parameters. Each referenced data object contains data that is independent from the request thus the same data object may be encountered across several responses with a different reference ID but with the same data.

Example:

```
{
    "query": {
        "domain": "zeroguard.com"
    },
    "data": {
        "records": [
            {
                "_ref": 1,
```

(continues on next page)

(continued from previous page)


```
                "ipv4": [
                    {
                        "_ref": 2,
                        "latest": true,
                        "oldest": false,
                        "live": false,
                        "seen_at": [
                            1584660099,
                            1584650121
                        ]
                    }
                ]
            }
        ],
        "references": {
            "1": {
                "type": "subdomain",
                "data": {
                    "name": "www.zeroguard.com"
                }
            },
            "2": {
                "type": "ipv4",
                "data": {
                    "address": "157.7.107.64",
                    "reputation": []
                }
            }
        }
    }
}
```

CHAPTER 8

Response Object Structure

All responses returned by ZeroGuard Recon API have a unified structure. This page describes how non-error responses are structured. For a structure of error responses refer to *Errors* section.

8.1 Structure

`Non-Error Response`
General structure of a non-error API response.

Object Properties

- **`_meta`** (*`object`*) – Contextual meta information about the response. May be absent.
- **`query`** (*`Query Object`*) – Contextual information about the query that was performed. Essentially, this object just mirrors request parameters that were specified (after a disambiguation and resolution process). Object structure is defined individually for each API endpoint. Always present.
- **`data`** (*`Data Object`*) – Results of a performed query (request). Always present.

`Query Object`
Structure of a query object.

Object Properties

- **`action`** (*`string`*) –

Todo: Fill out response body query object structure

`Data Object`
General structure of a non-error API response data.

Object Properties

- **records** (*list*) – List of records returned by the back-end. Object structure is defined individually for each API response. Always present.
- **references** (*map*) – Mapping of *_ref* (reference number) to a referenced object. See *Object References* for details. Included only if records reference any related objects. May be absent.

Todo: Describe a structure of *_meta* object

8.2 Examples

```
{
    "_meta": {},
    "query": {
        "domain": "example.com"
    },
    "data": {
        "records": [
        ],
        "references": {
        }
    }
}
```

A recommended way of getting data out of API response:

1. Check HTTP status code of the response.

Todo: More decent examples (or at least one example that is full)

CHAPTER 9

Search Subdomains

This endpoint allows to search for subdomains of a given domain. Search results will not only include subdomains but also all related to them objects (i.e. IP addresses, ASNs, RIRs).

Note: Please make sure you're familiar with *Data Types*, *Errors*, *Response Object Structure* and other general concepts as all information from these sections is directly applicable to this endpoint.

9.1 Schema

POST /v1/subdomains/(**str:** *domain*)

Search for all seen subdomains and related objects for *domain*.

All JSON parameters are prefixed with **<ip_version>** which determines a version of IP addresses on which a given JSON parameter acts on. This can be one of the following: **ipv4**, **ipv6**, **ip** (both versions). More specific version always prevails meaning that below parameters

```
{
    "ipv4.history": true,
    "ip.history": false
}
```

are resolved as

```
{
    "ipv4.history": true,
    "ipv6.history": false
}
```

Parameters

- **domain** (*str*) – Domain which subdomains to return.

JSON Parameters

- **<ip_version>.history** (*bool*) – Return a list of IP addresses to which a found subdomain pointed in the past. Default is *false*.
- **<ip_version>.latest** (*bool*) – Return the latest found IP address for each found subdomain. Default is *true*.
- **<ip_version>.oldest** (*bool*) – Return the oldest found IP address for each found subdomain. Default is *false*.
- **<ip_version>.live** (*bool*) – Perform a live DNS query and return its results for each found subdomain. Default is *false*.
- **<ip_version>.meta** (*bool*) – Return extra information about IP addresses related to each found subdomain. Default is *true*.

Status Codes

- 200 OK – Subdomains search was successfull.
- 400 Bad Request – Bad request. XXXXXXXFIXME See *Errors* for a general structure of an error response and *Response Data* for details on how error context is structured for this endpoint.
- 404 Not Found – Subdomains search was performed successfully but yielded no results.
- 429 Too Many Requests – API rate limit was exceeded. See *Rate Limits* for more information on how to gracefully handle API quota.
- 500 Internal Server Error – Internal server error. This is most probably a bug. See *Bugs Reporting* for more information about how to report bugs and security vulnerabilities.

Minimal Example:

Todo: Fully document a complex request example for */v1/subdomains* endpoint

---

http

```
POST /v1/subdomains/google.com HTTP/1.1
Host: api.zeroguard.com
Accept: application/json
```

curl

```
curl -i -X POST https://api.zeroguard.com/v1/subdomains/google.com -H 'Accept:␣
↪application/json'
```

wget

```
wget -S -O- https://api.zeroguard.com/v1/subdomains/google.com --header='Accept:␣
↪application/json'
```

python-requests

```
requests.post('https://api.zeroguard.com/v1/subdomains/google.com', headers={'Accept
↪': 'application/json'})
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json

{"stab": "StuB"}
```

Complex Example:

Todo: Fully document a complex request example for */v1/subdomains* endpoint

http

```
POST /v1/subdomains/complex?fields=foo%2Cbar%2Cbaz HTTP/1.1
Host: api.zeroguard.com
Accept: application/json
```

curl

```
curl -i -X POST 'https://api.zeroguard.com/v1/subdomains/complex?fields=foo%2Cbar
↪%2Cbaz' -H 'Accept: application/json'
```

wget

```
wget -S -O- 'https://api.zeroguard.com/v1/subdomains/complex?fields=foo%2Cbar%2Cbaz' -
↪-header='Accept: application/json'
```

python-requests

```
requests.post('https://api.zeroguard.com/v1/subdomains/complex?fields=foo%2Cbar%2Cbaz
↪', headers={'Accept': 'application/json'})
```

9.2 Response Data

Todo: Fully document a response data object structure for */v1/subdomains* endpoint

9.3 Error Context

Todo: Document error context for */v1/subdomains* endpoint

CHAPTER 10

Bugs Reporting

Please report any issues with this documentation or with a public API in general to bugs@zeroguard.com. For responsible disclosure see *Security Policy* section.

10.1 Security Policy

Todo: Fill out security policy file

This document is under construction

CHAPTER 11

Changelog

Note: Please note that this changelog includes both changes to the public API and changes to a documentation. Not every version of documentation indicates changes in the public API.

11.1 Version 0.0.1-dev1

Unreleased

- Work in progress version of documentation for APIv1.

CHAPTER 12

Roadmap

To be published.

Todo: Fill out a roadmap page

CHAPTER 13

To Do

This is a list of TODOs that cover all missing bits and pieces that are to be filled out in the future.

Todo: Fill out security policy file

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/zeroguard-api-docs/checkouts/latest/docs/about/bugs.rst, line 14.)

Todo: Fill out a roadmap page

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/zeroguard-api-docs/checkouts/latest/docs/about/roadmap.rst, line 6.)

Todo: Fill out an authentication documentation page

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/zeroguard-api-docs/checkouts/latest/docs/concepts/authentication.rst, line 5.)

Todo: Replace IPv4 address data type example with an actual output from the API. Currently it is a stub.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/zeroguard-api-docs/checkouts/latest/docs/concepts/data-types.rst, line 69.)

Todo: Replace IPv6 address data type example with an actual output from the API. Currently it is a stub.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/zeroguard-api-docs/checkouts/latest/docs/concepts/data-types.rst, line 141.)

Todo: Replace Network Prefix data type stub with an actual definition

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/zeroguard-api-docs/checkouts/latest/docs/concepts/data-types.rst, line 182.)

Todo: Propertly document Rate Limit Exceeded error type

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/zeroguard-api-docs/checkouts/latest/docs/concepts/errors.rst, line 301.)

Todo: Fill out rate limits documentation

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/zeroguard-api-docs/checkouts/latest/docs/concepts/rate-limits.rst, line 5.)

Todo: Object references explanation sucks. Make it better.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/zeroguard-api-docs/checkouts/latest/docs/concepts/references.rst, line 5.)

Todo: Fill out response body query object structure

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/zeroguard-api-docs/checkouts/latest/docs/concepts/response-structure.rst, line 38.)

Todo: Describe a structure of *_meta* object

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/zeroguard-api-docs/checkouts/latest/docs/concepts/response-structure.rst, line 56.)

Todo: More decent examples (or at least one example that is full)

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/zeroguard-api-docs/checkouts/latest/docs/concepts/response-structure.rst, line 84.)

Todo: Fully document a complex request example for */v1/subdomains* endpoint

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/zeroguard-api-docs/checkouts/latest/docs/endpoints/subdomains.rst, line 80.)

Todo: Fully document a complex request example for */v1/subdomains* endpoint

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/zeroguard-api-docs/checkouts/latest/docs/endpoints/subdomains.rst, line 98.)

Todo: Fully document a response data object structure for */v1/subdomains* endpoint

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/zeroguard-api-docs/checkouts/latest/docs/endpoints/subdomains.rst, line 114.)

Todo: Document error context for */v1/subdomains* endpoint

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/zeroguard-api-docs/checkouts/latest/docs/endpoints/subdomains.rst, line 123.)

Todo: Fill out a features overview page (this one)

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/zeroguard-api-docs/checkouts/latest/docs/introduction/features.rst, line 5.)

HTTP Routing Table

/v1

`POST /v1/subdomains/(str:domain),25`

Index

J