



# Exporting access logs to a CSV file

## ON THIS PAGE

[About the Access Logs API](#)[Identifying the accessed resources](#)[Rate limits](#)[Limitations](#)[What you'll need](#)[Creating the export script](#)[Exporting access logs with the script](#)[Modifying the script](#)[Updating the authentication credentials](#)[Adding admins](#)[Filtering the logs by other identifiers](#)[Getting additional details about the accessed tickets or users](#)

Access logs give you a picture of who is accessing what in your account. The access logs track the last 90 days of access activity by admins and agents but not end users. The Access Logs API allows you to export the access logs.

Access logs drive accountability by identifying data security risks, refining security and privacy policies, and supporting data privacy compliance. For example, you can use the data provided by the access logs to establish proper permissions for your agents with custom agent roles. See [Creating custom roles and assigning agents](#) in Zendesk help.

Access logs are only available with the [Zendesk Advanced Data Privacy and Protection add-on](#).

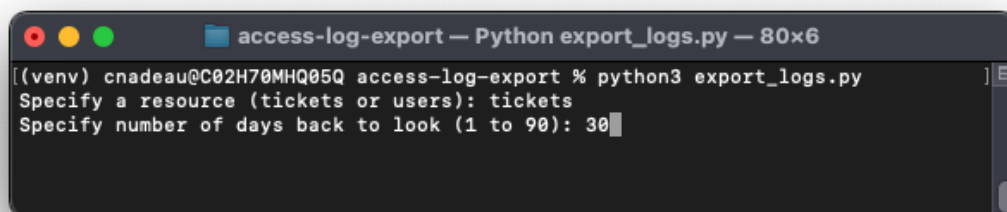
This article provides a demo script that answers the following questions:

- What tickets are agents accessing?

- What user profiles are agents accessing?

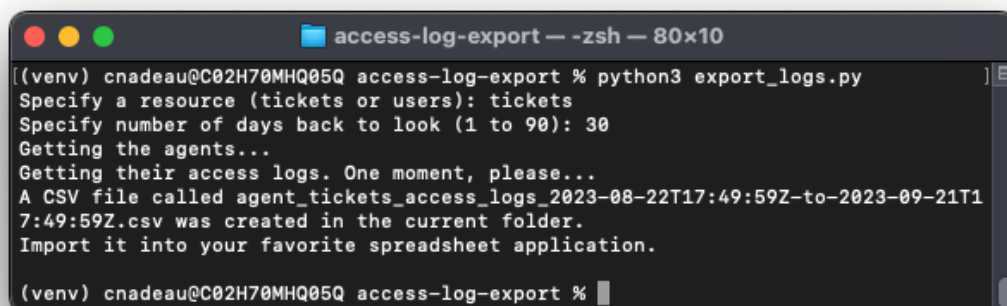
The script uses the [Access Logs API](#) to export the agents' access logs to a CSV (comma-separated values) file for further analysis.

The script asks the user for a resource (tickets or users) and the number of days back to look for logs:



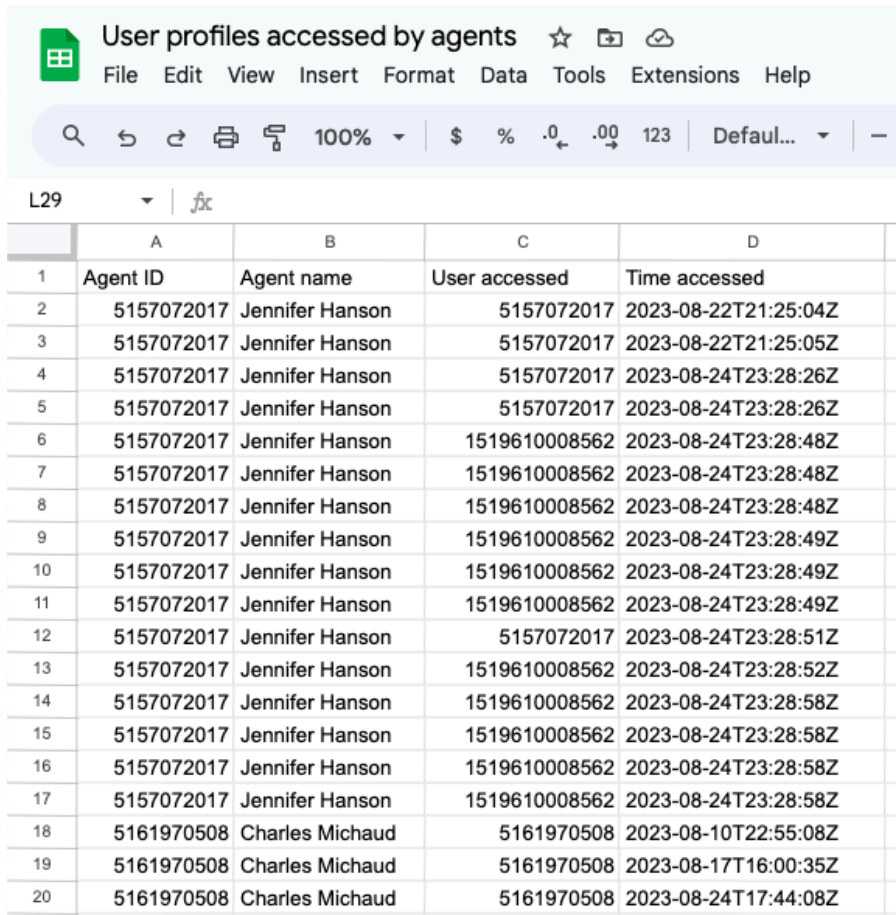
```
access-log-export — Python export_logs.py — 80x6
((venv) cnadeau@C02H70MHQ05Q access-log-export % python3 export_logs.py
Specify a resource (tickets or users): tickets
Specify number of days back to look (1 to 90): 30
```

The script gets the agents in the account, retrieves their access logs, filters them for the specified resource, and writes a CSV file:



```
access-log-export — -zsh — 80x10
((venv) cnadeau@C02H70MHQ05Q access-log-export % python3 export_logs.py
Specify a resource (tickets or users): tickets
Specify number of days back to look (1 to 90): 30
Getting the agents...
Getting their access logs. One moment, please...
A CSV file called agent_tickets_access_logs_2023-08-22T17:49:59Z-to-2023-09-21T17:49:59Z.csv was created in the current folder.
Import it into your favorite spreadsheet application.
((venv) cnadeau@C02H70MHQ05Q access-log-export %
```

Then you can import the CSV file in any spreadsheet application:



	A	B	C	D
1	Agent ID	Agent name	User accessed	Time accessed
2	5157072017	Jennifer Hanson	5157072017	2023-08-22T21:25:04Z
3	5157072017	Jennifer Hanson	5157072017	2023-08-22T21:25:05Z
4	5157072017	Jennifer Hanson	5157072017	2023-08-24T23:28:26Z
5	5157072017	Jennifer Hanson	5157072017	2023-08-24T23:28:26Z
6	5157072017	Jennifer Hanson	1519610008562	2023-08-24T23:28:48Z
7	5157072017	Jennifer Hanson	1519610008562	2023-08-24T23:28:48Z
8	5157072017	Jennifer Hanson	1519610008562	2023-08-24T23:28:48Z
9	5157072017	Jennifer Hanson	1519610008562	2023-08-24T23:28:49Z
10	5157072017	Jennifer Hanson	1519610008562	2023-08-24T23:28:49Z
11	5157072017	Jennifer Hanson	1519610008562	2023-08-24T23:28:49Z
12	5157072017	Jennifer Hanson	5157072017	2023-08-24T23:28:51Z
13	5157072017	Jennifer Hanson	1519610008562	2023-08-24T23:28:52Z
14	5157072017	Jennifer Hanson	1519610008562	2023-08-24T23:28:58Z
15	5157072017	Jennifer Hanson	1519610008562	2023-08-24T23:28:58Z
16	5157072017	Jennifer Hanson	1519610008562	2023-08-24T23:28:58Z
17	5157072017	Jennifer Hanson	1519610008562	2023-08-24T23:28:58Z
18	5161970508	Charles Michaud	5161970508	2023-08-10T22:55:08Z
19	5161970508	Charles Michaud	5161970508	2023-08-17T16:00:35Z
20	5161970508	Charles Michaud	5161970508	2023-08-24T17:44:08Z

**Disclaimer:** Zendesk provides the script for illustrative purposes only. Zendesk does not support or guarantee the code in the script. Zendesk also can't provide support for third-party technologies such as Python.

## About the Access Logs API

The Access Logs API lets admins export up to the last 90 days of account access activity by team members. Team members include both admins and agents but not end users.

The API lets you filter the logs by a time period, a specific user, or a specific resource, such as tickets. You can also use scripting to further filter the data returned.

Each log represents an access event and includes the following information:

- an identifier of the resource that was accessed (see [Identifying the accessed resources](#))
- the id of the user who accessed the resource
- the timestamp of when the user accessed the resource.

The log also includes other related information. See [Access Logs](#).

Example access log:

```
1  {
2    "id": "01H7TC9307Z5S07QPRW8REK5RD",
3    "ip_address": "52.40.158.85",
4    "method": "GET",
5    "origin_service": "Zendesk",
6    "status": 200,
7    "timestamp": "2023-08-14T15:57:48Z",
8    "url": "/api/v2/users/417198039213/{...}",
9    "user_id": 364425201147
10 }
```

In the example, a team member with the id of 364425201147 accessed the profile of a user with the id of 417198039213, as identified by the `url` property. The team member accessed the resource on 2023-08-14 at 15:57:48 UTC.

## Identifying the accessed resources

The access logs indirectly identifies the resource that a user accessed by providing one of the following identifiers:

- the URL path of the REST API endpoint used to access the resource
- the GraphQL query used to access the resource

Many resources in the Zendesk frontend are accessed by REST APIs. REST APIs use uniform resource identifiers (URIs) to address resources. If a REST API was used to access a Zendesk resource, the `url` property in the access log will specify a URI that identifies the resource. Example: `"url": "/api/v2/users/417198039213"`. In this example, the user accessed the profile page of the user with the id of 417198039213. To learn more about API URIs, see [REST API URI Naming Conventions and Best Practices](#) on [restfulapi.net](#).

Other parts of the Zendesk frontend are accessed using GraphQL, a query language for APIs. If a GraphQL query was used to access a resource, the access log will specify `"/graphql"` as the value of the `url` property. In addition, the log will include a `graphql` object with the following properties: `operation_name`, `operation_type`, `query`, and `variables`. All other log information is the same.

The `query` property may specify something like `"query ticket"` or `"query user"` and the `variables` property will specify the `id` of the ticket or user to look up.

### REST API identifier example

```
1  {
2    "id": "01H7TC9307Z5S07QPRW8REK5RD",
3    "ip_address": "52.40.158.85",
4    "method": "GET",
5    "origin_service": "Zendesk",
```

```

6      "status": 200,
7      "timestamp": "2023-09-16T19:00:00Z",
8      "url": "/api/v2/search/incremental?{...}&query=agent guidelines",
9      "user_id": 364425201147
10    },

```

In the example, the url indicates that the user searched for "agent guidelines".

### GraphQL identifier example

```

1  {
2    "graphql": {
3      "operation_name": "ticket",
4      "operation_type": "QUERY",
5      "query":
6      "\"query ticket($id: ID!, $includeSkills: Boolean = false, $includeCustomFields: Boolean = false, $includeConversationAuthenticated: Boolean = false) { ticket(id: $id) { id assigne
7    },
8    "id": "01H7TC933PWS5B33SS5QHYZVKB",
9    "ip_address": "52.40.158.85",
10   "method": "POST",
11   "origin_service": "Zendesk",
12   "status": 200,
13   "timestamp": "2023-08-14T15:57:49Z",
14   "url": "/graphql",
15   "user_id": 364425201147
16 },

```

In the example, the query property specifies "query ticket (\$id: ID!...), ...". The \$id is a variable and its value is specified in the variables property as "{\\"id\\":\\"199\\", ...". This means the user accessed a ticket with the id of 199.

## Rate limits

Requests are rate limited to 50 requests per minute, including pagination requests. Accordingly, you should use the maximum page size of 2500 to get the most records before reaching the limit.

The Access Logs API doesn't follow the same rate limiting conventions as other Zendesk APIs. Responses don't include x-rate-limits and x-rate-limit-remaining headers. The API does return a 429 status code when the limit is reached but doesn't include a retry-after header. You should use a "retry after" value of 60 seconds in your code.

# Limitations

The Access Logs API has the following limitations:

- The API only indirectly identifies a resource that an agent accessed. The identifier is either the URL of the API endpoint used to access the resource or the GraphQL query used to access the resource. A basic knowledge of REST APIs and GraphQL queries is required to interpret the data.
- If you move your account to another [pod](#) (point of delivery), you'll lose access to the access information in the old pod. This data is not copied over.
- There are still some parts of the product where access information is not gathered. Examples include Explore and Sell.

## What you'll need

This article provides a script that uses the Access Logs API to export the access logs to a CSV (comma-separated values) file. To run the script in this article, you'll need the following:

- A Zendesk account

You'll need admin permissions to a Zendesk account with the [Zendesk Advanced Data Privacy and Protection add-on](#).

- Python

The script in this article uses the Python programming language. To install the latest version of Python, see <http://www.python.org/download/>.

- Requests library for Python

The Python script uses the [Requests library](#) for Python. The Requests library simplifies making API requests in Python. To install it, make sure you install Python first and then run the following command in your terminal:

```
1 pip3 install requests
```

- Arrow library for Python

The Python script uses the [Arrow library](#) for Python. The Arrow library provides a friendlier, more streamlined way of working with dates and times. To install it, make sure you install Python first and then run the following command in your terminal:

```
1 pip3 install arrow
```

**Note:** Some lines of code in the examples may wrap to the next line because of the article's page width. When copying the script, ignore the line wrapping. Line breaks matter in Python.

# Creating the export script

1. Create a folder called **access-log-export** on your system.
2. In a plain text editor, create a file named **export\_logs.py** and save it in the folder.
3. Paste the following code in the file:

```

1  import json
2  from csv import writer
3  from pathlib import Path
4  from time import sleep
5  import re
6
7  import requests
8  import arrow
9
10 import os
11
12 """
13 Specify your Zendesk subdomain and credentials. In production, use
   environment variables instead.
14
15 """
16 ZENDESK_SUBDOMAIN = 'yoursubdomain'
17 ZENDESK_USER_EMAIL = 'youremail@example.com'
18 ZENDESK_API_TOKEN = os.getenv('ZENDESK_API_TOKEN')
19
20 def export_agent_access_logs() -> list:
21     """
22     Asks the user for a resource (tickets or users) that agents have
   accessed in the last number of days. Exports the results to a CSV file.
23
24     :return: A list of tickets or users that agents accessed, the agents
   that accessed them, and when they accessed them
25
26     """
27
28     # -- Ask for the resource and the number of days ----- #
29
30     while True:
31         resource = input('Specify a resource (tickets or users): ')
32         if resource in ['users', 'tickets']:
33             break
34         else:
35             print('Not a valid option. Try again.')
36
37     while True:
38         days = input('Specify number of days back to look (1 to 90): ')

```

```

37         if days.isdigit() and 1 <= int(days) <= 90:
38             days = int(days)
39             break
40         print('Not a valid option. Try again.')
41
42     # -- Get agents ----- #
43
44     print('Getting the agents...')
45     agents = []
46     api_list_name = 'users'
47     url = f'https://{ZENDESK_SUBDOMAIN}.zendesk.com/api/v2/users'
48     params = {'page[size]': 100}
49     users = get_api_list(api_list_name, url, params)
50     for user in users:
51         if user['role'] == 'agent':
52             agents.append(user)
53
54     # -- Get agent access logs ----- #
55
56     print('Getting their access logs. One moment, please...')
57     agent_access_logs = []
58     api_list_name = 'access_logs'
59     url = f'https://{ZENDESK_SUBDOMAIN}.zendesk.com/api/v2/access_logs'
60     end_time = arrow.utcnow()
61     filter_end = end_time.format('YYYY-MM-DD[T]HH:mm:ss[Z]')
62     start_time = end_time.shift(days=-days)
63     filter_start = start_time.format('YYYY-MM-DD[T]HH:mm:ss[Z]')
64
65     params = {
66         'filter[start]': filter_start,
67         'filter[end]': filter_end,
68         'page[size]': 1000
69     }
70     for agent in agents:
71         params['filter[user_id]'] = agent['id']
72         access_logs = get_api_list(api_list_name, url, params)
73         for log in access_logs:
74             log['agent_name'] = agent['name']
75         agent_access_logs.extend(access_logs)
76
77     # -- Filter the logs by the specified resource ----- #
78
79     agent_resource_access_logs = []
80     identifiers = {
81         'tickets': {'api': '/api/v2/tickets/', 'graphql': 'query ticket'},
82         'users': {'api': '/api/v2/users/', 'graphql': 'query user'}
83     }
84     resource_identifiers = identifiers[resource]
85     for log in agent_access_logs:

```



```

86
87     # look for the resource in the api request
88     if 'api' in resource_identifiers and resource_identifiers['api'] in
log['url']:
89         match = re.search(r'\d{2,}', log['url'])
90         if match:
91             log['resource_id'] = match.group()
92             agent_resource_access_logs.append(log)
93
94     # look for the resource in the graphql query
95     elif 'graphql' in log:
96         if 'graphql' in resource_identifiers and resource_identifiers[
'graphql'] in log['graphql']['query']:
97             variables = json.loads(log['graphql']['variables'])
98             if 'id' in variables:
99                 log['resource_id'] = str(variables['id'])
100                 agent_resource_access_logs.append(log)
101
102     if not agent_resource_access_logs:
103         print(f'Agents did not access any {resource} in the last {days}
days.')
104         return []
105
106     # -- Create the CSV file ----- #
107
108     resource_name = resource[:-1].capitalize()
109     rows = [
110         (
111             f'Agent ID',
112             'Agent name',
113             f'{resource_name} accessed',
114             'Time accessed'
115         )
116     ]
117     for log in agent_resource_access_logs:
118         row = (
119             log['user_id'],
120             log['agent_name'],
121             log['resource_id'],
122             log['timestamp']
123         )
124         rows.append(row)
125
126     file_path = Path(f'agent_{resource}_access_logs_{filter_start}-to-{
filter_end}.csv')
127     with file_path.open(mode='w', newline='') as csv_file:
128         report_writer = writer(csv_file, dialect='excel')
129         for row in rows:
130             report_writer.writerow(row)

```

```

131
132     print(f'A CSV file called {file_path}
was created in the current folder.\n'
133           f'Import it into your favorite spreadsheet application.\n')
134
135     return agent_resource_access_logs
136
137
138 # -- API request function ----- #
139
140 def get_api_list(api_list_name, url, params) -> list:
141     """
142     Makes an API request to a Zendesk list endpoint and returns a list of
results.
143
144     :param str api_list_name: The name of the list returned by the API.
Examples: 'access_logs' or 'users'. See the reference docs for the name
145
146     :param str url: The endpoint url.
147     :param dict params: Query parameters for the endpoint.
148     :return: List of resource records.
149     """
150     api_list = []
151     # Zendesk API token usage in the format 'email/token:api_token'
152     auth = f'{ZENDESK_USER_EMAIL}/token', ZENDESK_API_TOKEN
153     while url:
154         response = requests.get(url, params=params, auth=auth)
155         if response.status_code == 429:
156             if 'retry-after' in response.headers:
157                 wait_time = int(response.headers['retry-after'])
158             else:
159                 wait_time = 60
160             print(f'Rate limited! Please wait. Will restart in {wait_time}
seconds.')
161             sleep(wait_time)
162             response = requests.get(url, params=params, auth=auth)
163
164         if response.status_code != 200:
165             print(f'Error -> API responded with status {response.status_code
}: {response.text}. Exiting.')
166             exit()
167
168         data = response.json()
169         api_list.extend(data[api_list_name])
170         if data['meta']['has_more']:
171             params['page[after]'] = data['meta']['after_cursor']
172         else:
173             url = ''

```

```
173         return api_list
174
175
176     if __name__ == '__main__':
177         export_agent_access_logs()
```

4. Save the file.

## Exporting access logs with the script

Before running the script for the first time, update the values of the variables at the top of the file:

```
1  ZENDESK_SUBDOMAIN = 'yoursubdomain'
2  ZENDESK_USERNAME = 'youremail@example.com'
3  ZENDESK_TOKEN = os.getenv('ZENDESK_API_TOKEN')
```

Follow best practices for handling API tokens and never hard-code your API token within your source code. Here, we define the environment variable `ZENDESK_API_TOKEN` to contain the token.

### To run the script

1. In your terminal, navigate to your **access-log-export** folder.
2. Enter the following command and press Enter.

```
1  python3 export_logs.py
```

3. Follow the prompts.

The script creates a CSV file in your **access-log-export** folder. Import the CSV file into a spreadsheet application.

## Modifying the script

If you want, you can modify the script. This section describes the different parts of the script so you have a better understanding of how it works before making changes to it.

## Updating the authentication credentials

The script uses API tokens, which only requires the user name and API token of a Zendesk admin:

```
1  ZENDESK_USERNAME = 'jdoe@example.com'
```

```
2 ZENDESK_TOKEN = os.getenv('ZENDESK_API_TOKEN')
```

You should use environment variables in production. In Python, you can retrieve the environment variables as follows:

```
1 import os
2
3 ZENDESK_USERNAME = os.getenv('ZEN_USER')
4 ZENDESK_TOKEN = os.getenv('ZENDESK_API_TOKEN')
```

You can also use other authentication methods such as an OAuth token. For more information, see [Security and authentication](#).

## Adding admins

In addition to agents, you could include the access logs of admins. When getting the agents from the Users API, include users with the role of admin too:

```
1 for user in users:
2     if user['role'] == 'agent' or user['role'] == 'admin':
3         agents.append(user)
```

## Filtering the logs by other identifiers

You can add other identifiers in addition to the ones for tickets and users. In the "Filter the logs by the specified resource" section of the script, add the new resource's identifiers in the `identifiers` dictionary. The following example adds ticket forms:

```
1 identifiers = {
2     'tickets': {'api': '/api/v2/tickets/', 'graphql': 'query ticket'},
3     'users': {'api': '/api/v2/users/', 'graphql': 'query user'},
4     'ticket_forms': {'api': '/api/v2/ticket_forms/', 'graphql':
5         'query ticket_forms'}
6 }
```

Then modify the API and GraphQL filters as necessary to handle the new identifiers.

Finally, add a new option that the user can enter when they start the script:

```
1 resource = input('Specify a resource (tickets, users, or ticket_forms): ')
2 if resource in ['users', 'tickets', 'ticket_forms']:
```

3

`break`

## Getting additional details about the accessed tickets or users

You can make other requests to the Zendesk API to get additional details about the tickets or users that agents accessed and then add the information to your CSV file. See [Show Ticket](#) or [Show User](#) in the API reference.

For example, after populating the `agent_resource_access_logs` variable in the script, you could add the following `for` loop to iterate over all the accessed tickets or users and make a request for the record for each:

```
1  for log in agent_resource_access_logs:
2      url = f'https://{ZENDESK_SUBDOMAIN}.zendesk.com/api/v2/{resource}/{log[
    "resource_id"]}'
3      data = get_api_resource(url)
4      log['resource_record'] = data
5
6      # ... add additional details to the CSV file
```

The loop uses a different function to make the API requests. The `get_api_list()` function in the main script wouldn't work in this case because these endpoints only return a single record at a time and don't use pagination. Here's a possible function you could use to make requests for single records:

```
1  def get_api_resource(url):
2      """
3      Returns a single resource record.
4      :param url: A full endpoint url, such as
        'https://example.zendesk.com/api/v2/tickets/12345
5
6      :return: The specified record of the resource
7      """
8
9      auth = f'{ZENDESK_USER_EMAIL}/token', ZENDESK_API_TOKEN
10
11     response = requests.get(url, auth=auth)
12     if response.status_code == 429:
13         print('Rate limited! Please wait.')
14         sleep(int(response.headers['retry-after']))
15         response = requests.get(url, auth=auth)
16     if response.status_code != 200:
17         print(f'Error -> API responded with status {response.status_code}: {
            response.text}. Exiting.')
18         exit()
19
20     # return only the resource data, not the name of the resource
```