```
52451 collector_down
52450 collector_down
52449 collector_down
52448 collector_down
52447 collector_down
52446 collector_down
52445 collector_down
52444 collector_down
Getting alerts for the mssp_user account
52414 dos_host_detection IP Fragmentation, Total Traffic, UDP
52405 dos_host_detection TCP SYN, Total Traffic
52404 dos_host_detection TCP SYN, Total Traffic
52401 dos_host_detection TCP SYN, Total Traffic
52400 dos_host_detection TCP SYN, Total Traffic
52313 dos_host_detection TCP SYN, Total Traffic
52312 dos_host_detection TCP SYN, Total Traffic
52309 dos_host_detection TCP SYN, Total Traffic
52308 dos_host_detection TCP SYN, Total Traffic
52220 dos_host_detection TCP SYN, Total Traffic
52219 dos_host_detection TCP SYN, Total Traffic
52216 dos_host_detection TCP SYN, Total Traffic
52215 dos_host_detection TCP SYN, Total Traffic
52203 dos_host_detection IP Fragmentation, Total Traffic, UDP
52127 dos_host_detection TCP SYN, Total Traffic
```

The system alerts are only available to the administrative account, and the MSSP user account only sees the `dos_host_detection` alerts for the managed objects it is scoped to.

Because subobjects of the results of endpoints are not yet scoped, if any part of the results from an endpoint are restricted from view by a non-administrative user, the entirety of the results of the endpoint are not displayed, and instead a `404 NOT FOUND` HTTP status code is returned. This behavior extends to use of the `?include` query parameter; if your API query requests the inclusion of data from an endpoint it is not authorized to access, the entire query will result in a `404 NOT FOUND` HTTP status code and no data will be returned.

This is documented in more detail at `https://spleader.example.com/api/sp/doc/v4/endpoints.html#authorization`.

Using the SP REST API to write a client that supports caching (or Why the Sub-Object JSON is so Deeply Nested)

The SP REST API is written so that its output for objects that have multiple sub-endpoints can be combined without having to restruc-

ture the data that you already have or the new data that was just returned. This is useful when your client needs to cache data or wants to create data sets by assembling the relevant data from API sub-endpoints. We aren't going to provide a full example of a caching client in this section, but we will describe how this could be done using the SP REST API and you will see how the data from the sub-endpoints fits into the JSON object that comes from the top-level endpoint.

In SP 8.4 the endpoint that has the most sub-endpoints is the `/alerts/` endpoint, so we will use that as the example. Starting with the alert having ID 172784 (and trimming out some of the response in the interest of space) the API request response for that alert is


```
{
  "data": {
    "attributes": {
      "alert_class": "dos",
      "alert_type": "dos_profiled_router",
      "ongoing": false,
      "start_time": "2018-07-04T01:04:00+00:00",
      "stop_time": "2018-07-04T01:28:36+00:00",
      "subobject": {
        "direction": "Outgoing",
        "fast_detected": false,
        "impact_bps": 6376742,
        "impact_pps": 1883,
        "ip_version": 4,
        "protocols": [],
        "severity_percent": 127.53,
        "severity_threshold": 5000000.0,
        "severity_unit": "bps"
      }
    },
    "id": "172784",
    "relationships": {
      "packet_size_distribution": {
        "data": {
          "id": "packet-size-distribution-172784",
          "type": "alert_packet_size_distribution"
        },
        "links": {
          "related":
          ↪ "https://spleader.example.com/api/sp/v5/alerts/172784/packet_size_distribution"
        }
      },
      "patterns": {
        "links": {
          "related":
          ↪ "https://spleader.example.com/api/sp/v5/alerts/172784/patterns/"
        }
      },
      "router_traffic": {
        "links": {
          "related":
          ↪ "https://spleader.example.com/api/sp/v5/alerts/172784/router_traffic/"
        }
      },
      "source_ip_addresses": {
        "data": {
          "id": "source-ip-addresses-172784",
          "type": "alert_source_ip_addresses"
        },
        "links": {
          "related":
          ↪ "https://spleader.example.com/api/sp/v5/alerts/172784/source_ip_addresses"
        }
      },
      "thresholds": {
```

```
          "links": {
            "related":
            ↪  "https://spleader.example.com/api/sp/v5/alerts/172784/interface_traffic_thresholds/"
          }
        },
        "traffic": {
          "data": {
            "id": "alert-traffic-172784",
            "type": "alert_traffic"
          },
          "links": {
            "related":
            ↪  "https://spleader.example.com/api/sp/v5/alerts/172784/traffic"
          }
        }
      },
      "type": "alert"
    }
}
```

The relationships shown in that output are all sub-objects of the `/alerts/172784` object. Taking `source_ip_addresses` as an example, the API response for that object (again, with some data trimmed) looks like

```
{
    "data": {
        "attributes": {
            "source_ips": [
                "130.182.0.0",
                "130.182.0.1",
                "130.182.0.2",
                "130.182.0.3",
                "130.182.0.4",
                "130.182.0.5",
                "130.182.0.6",
                "130.182.0.7",
                "130.182.0.8",
                "130.182.0.9"
            ]
        },
        "id": "source-ip-addresses-172784",
        "links": {
            "self":
            ↪  "https://spleader.example.com/api/sp/v5/alerts/172784/source_ip_addresses"
        },
        "relationships": {
            "parent": {
                "data": {
                    "id": "172784",
                    "type": "alert"
                },
                "links": {
                    "related":
                    ↪  "https://spleader.example.com/api/sp/v5/alerts/172784"
                }
            }
        },
        "type": "alert_source_ip_addresses"
    }
}
```

A more complicated example is one of the sub-endpoints that has more than one object, `router_traffic` (which has been greatly simplified so it can be seen):

```
{
    "data": [
```

```
        {
            "attributes": {
                "view": {
                    "router-245": {
                        "unit": {
                            "bps": {
                                "avg_value": 4386058,
                                "current_value": 815626,
                                "max_value": 6244499,
                                "pct95_value": 6019565,
                                "severity": 2,
                                "step": 60,
                                "timeseries": [
                                    5853386,
                                    5842086,
                                    6244499,
                                    5878551,
                                    5842066,
                                    5849164
                                ],
                                "timeseries_start": "2018-07-04T01:03:15+00:00"
                            }
                        }
                    }
                }
            },
            "id": "172784-245",
            "links": {
                "self":
                ↪  "https://spleader.example.com/api/sp/v5/alerts/172784/router_traffic/172784-2
            },
            "type": "alert_router_traffic"
        },
        {
            "attributes": {
                "view": {
                    "router-246": {
                        "unit": {
                            "bps": {
                                "avg_value": 4363147,
                                "current_value": 874213,
                                "max_value": 6290026,
                                "pct95_value": 6178240,
                                "severity": 2,
                                "step": 60,
                                "timeseries": [
                                    5948880,
                                    5504565,
                                    5907508,
                                    5786937,
                                    6061942,
                                    5686562
                                ],
                                "timeseries_start": "2018-07-04T01:03:15+00:00"
                            }
                        }
                    }
                }
            },
            "id": "172784-246",
            "links": {
                "self":
                ↪  "https://spleader.example.com/api/sp/v5/alerts/172784/router_traffic/172784-2
            },
            "type": "alert_router_traffic"
        }
    ]
}
```


Given those three pieces of data, the client starts by retrieving the alert information, but none of the related information. As the user of the client requests more details, they can be added to the data structure for the original alert and referenced after that. The pseudo-

code for this looks like

```
if 'user_requested_key' is not in results['alert_id'] then:
        make_api_request_for['user_requested_key']for['alert_id']
endif

return results['alert_id']['user_requested_key']
```

which means the first request for any particular data in a sub-endpoint will be slightly slower as the API request is made, but any subsequent request will already be in memory. Of course, you could add information about how old the data is and refresh the data using API calls if it is older than your threshold.

Using the example results data from above we can use pseudo-code that looks like this (and only handles one level of depth)

```
for key in results_from_source_ip_addresses['data']['attributes']:
    do:
        results['data']['attributes']['key'] =
            results_from_source_ip_addresses['data']['attributes']
```

to populate the original alert data results. This then looks like (with the `relationships` links removed for brevity) this


```
{
    "data": {
        "attributes": {
            "alert_class": "dos",
            "alert_type": "dos_profiled_router",
            "ongoing": false,
            "start_time": "2018-07-04T01:04:00+00:00",
            "stop_time": "2018-07-04T01:28:36+00:00",
            "subobject": {
                "direction": "Outgoing",
                "fast_detected": false,
                "impact_bps": 6376742,
                "impact_pps": 1883,
                "ip_version": 4,
                "protocols": [],
                "severity_percent": 127.53,
                "severity_threshold": 5000000.0,
                "severity_unit": "bps"
            },
            "source_ips": [
                "130.182.0.0",
                "130.182.0.1",
                "130.182.0.2",
                "130.182.0.3",
                "130.182.0.4",
                "130.182.0.5",
                "130.182.0.6",
                "130.182.0.7",
                "130.182.0.8",
                "130.182.0.9"
            ]
        },
        "id": "172784",
        "type": "alert"
    }
}
```


allowing for code that can handle more levels of depth than the pseudo-code above, the data from the `router_traffic` endpoint integrates into the alert data to result in

```
{
    "data": {
        "attributes": {
            "alert_class": "dos",
            "alert_type": "dos_profiled_router",
            "ongoing": false,
            "start_time": "2018-07-04T01:04:00+00:00",
            "stop_time": "2018-07-04T01:28:36+00:00",
            "subobject": {
                "direction": "Outgoing",
                "fast_detected": false,
                "impact_bps": 6376742,
                "impact_pps": 1883,
                "ip_version": 4,
                "protocols": [],
                "severity_percent": 127.53,
                "severity_threshold": 5000000.0,
                "severity_unit": "bps"
            },
            "source_ips": [
                "130.182.0.0",
                "130.182.0.1",
                "130.182.0.2",
                "130.182.0.3",
                "130.182.0.4",
                "130.182.0.5",
                "130.182.0.6",
                "130.182.0.7",
                "130.182.0.8",
                "130.182.0.9"
            ],
            "view": {
                "router-245": {
                    "unit": {
                        "bps": {
                            "avg_value": 4386058,
                            "current_value": 815626,
                            "max_value": 6244499,
                            "pct95_value": 6019565,
                            "severity": 2,
                            "step": 60,
                            "timeseries": [
                                5853386,
                                5842086,
                                6244499,
                                5878551,
                                5842066,
                                5849164
                            ],
                            "timeseries_start": "2018-07-04T01:03:15+00:00"
                        }
                    }
                },
                "router-246": {
                    "unit": {
                        "bps": {
                            "avg_value": 4363147,
                            "current_value": 874213,
                            "max_value": 6290026,
                            "pct95_value": 6178240,
                            "severity": 2,
                            "step": 60,
                            "timeseries": [
                                5948880,
                                5504565,
                                5907508,
                                5786937,
                                6061942,
                                5686562
                            ],
                            "timeseries_start": "2018-07-04T01:03:15+00:00"
                        }
                    }
                }
            }
```

```
        },
        "id": "172784",
        "type": "alert"
    }
}
```

Staring at JSON data doesn't always lead to clarity, but a close enough look at it shows that adding data to the original JSON output from the `/alerts/<alert_id>` endpoint is done with extraction of data from the sub-endpoints and direct insertion into the original results without any reformatting or refactoring of the data; the only real effort is knowing which level of data from the sub-endpoints should go into which level of the data from the top-level endpoint.

The structure of the data from the sub-endpoints might look unnecessarily nested or complicated when taken out of context, however, this structure allows for simple construction of the entirety of the alert data into one JSON object.