Supercharge your Malware Analysis Workflow with Assemblyline
The A(semblyline)-Team

- Steve Garon – Team Leader
- Kevin Hardy-Cooper – Dynamic Analysis
- Ryan Samaroo – Core Infrastructure
- Gabriel Desmarais – Services
- Marc-Olivier Guilbault – Dynamic Analysis

@ assemblyline@cyber.gc.ca
_discord.gg/GUAy9wErNu
Syllabus

- A Little Bit of History
- Design and Architecture
- What deployment works for you?
- User Interface Showcase
  
  Coffee Break

- Assemblyline API Walkthrough
  
  Lunch

- The different parts of a service
- Service Creation
  
  Coffee Break

- Service Creation (Wrap up)
- Scale your deployment
- Future Work

- 20 min (9:30 – 9:50)
- 1h (9:50 – 10:50)
- 1h 20min (11:20 – 12:40)
- 20min (14:00 – 14:20)
- 1h (14:20 – 15:20)
- 1h (15:50 – 16:50)
- 20min (16:50 – 17:10)
A Little Bit of History
... Back in the days

Small team of 3 Reverse Engineers

Daily Stats:
• ~ 10 files received
• ~ 5 Unique
• ~ Between 0 – 5 analysed
"I SPEND A LOT OF TIME ON THIS TASK. I SHOULD WRITE A PROGRAM AUTOMATING IT!"

**THEORY:**
- Work
- Work on Original Task
- Automation Takes Over
- Free Time

**REALITY:**
- Work
- Writing Code
- Debugging
- Ongoing Development
- No Time for Original Task Anymore

Source: https://xkcd.com/1319/
Surely we can do more?
Detect and respond to all malware targeting the Government of Canada
From a handful of files to millions

* just wave your magic scale wand *

Source: http://gunshowcomic.com/648
The Foundation for the Future – Assemblyline 3

- Distributed analysis platform
- Aggressive deduplication
- Alerting system with automated workflows
- Scalable
- Open source since Oct 2017

- About 2M files scanned daily
- Between 3K -150K alerts
- About 4500 files per minute during peak times
- Can only keep a week of data
Let’s start over but do it right this time...
Now we’re talking!

• No more backlogs
• 3.5M+ files after deduplication
• Up to 15K+ files per minute during peak times
• Currently keeping 2 months of data
  • 4.7 TB / 1.8 billion docs – Elastic index
• Icing on the cake: Not a single DB crash in the past 2 years!
  • Kudos to Elastic!
Ok stop stalling, how does this work?

Import your ruleset!

External systems

Antivirus
Sandboxes
Security products

- 50+ different services
- Generates score, heuristics, tags
- Recursively analyzes embedded elements

Email
Gateways
EDR
Analyst / IR / FORENSIC

Alerts
Indicators of compromise
Automatic tasking via Webhook / API
Who is it for?

- Government Provincial / Federal
- Corporate Organizations
- CERTs
- Malware research labs
- Academia
- InfoSec community

*NOT recommended* for personal use or to replace any desktop AV...
Design and Architecture
Core Components

- **Ingester**
  - High volume ingestion component

- **Dispatcher**
  - Core tasking component

- **Scaler**
  - Service load balancer component

- **Updater**
  - Service updater component
Core Components (continued…)

- **Service Server**
  - Separate the services from the core components

- **Expiry**
  - Removes expired data/files based on TTL (Time To Live)

- **Alerter**
  - Creates alerts using all the information about the submission when requirements are met

- **Workflow**
  - Auto label, prioritise and set status on alerts
UI Components

- **API Server**
  - Hosts the different API endpoints and makes sure access control is respected

- **Socket Server**
  - Hosts the WebSocket endpoints

- **Frontend**
  - Hosts all static JavaScript, HTML and image files used in the UI
Creating your deployment
Choosing the right deployment type

• How many files a setup can process depends on:
  • Size of files
  • Types of files
  • Types of analysis services
  • Number of services that you will be running
  • The quantity of resources those services use
• We can only offer very rough ideas
Our current biggest deployment

- Auto-scalable 12-72 nodes cluster (16 cores/64 GB per node)
- Up to 3.5M+ unique submissions a day (Avg. 1.5M)
- Lots of downtime during the night
- Rarely uses the full node capacity
- Mixed file types, mix of static and dynamic analysis.
Development VM using an IDE

- Throughput: A couple at a time
- Only includes components that play nice with virtualization
- Easy to set up
- Wants its own operating system
  - Needs a VM, not a container
- Like the name says, this is for development, nothing else
- Minimum resources: 2 cores / 6GB of ram

* https://cybercentrecanada.github.io/assemblyline4_docs/developer_manual/env/vscode/setup_script/*
Appliance (docker compose)

- Small throughput
- Few hundred per minute
- Fairly easy to set up
- Can be installed on a server or a VM
- Everything on the same box
- Not recommended to use with logging stack since everything is on the same box

* https://cybercentrecanada.github.io/assemblyline4_docs/installation/appliance/docker/
Cluster (Kubernetes)

- High Throughput
  - Up to tens of thousand files per minute
- Auto-scaling of all major components
  - Services
  - API endpoints
  - Core components
- Even number of nodes (VMs) auto-scales if deployed in supported environment (cloud)
- Logging stack recommended to keep track of the logs
- Cost a lot more, harder to setup

* https://cybercentrecanada.github.io/assemblyline4_docs/installation/cluster/general/
Using Assemblyline

Discover Assemblyline's User Interface
Live demo!
Automate all the things!
An introduction to the Assemblyline API
We have a test deployment ready for you...

- https://ec2-3-98-100-58.ca-central-1.compute.amazonaws.com

- Credentials
  - Username: first
  - Password: f1r$tD3m0p@ssw0rd!
**Introduction to Assemblyline API**

- Assemblyline uses REST APIs for system interaction
  - RBAC
  - Various means of authentication (Basic, API keys, OBO)

<table>
<thead>
<tr>
<th>Account Type</th>
<th>Administrator</th>
<th>User</th>
<th>Signature Manager</th>
<th>Signature Importer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Data Viewer</td>
<td>Submission Creator</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Roles</th>
<th>Administration</th>
<th>Alert Manage</th>
<th>Alert View</th>
<th>APIKey Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Archive Download</td>
<td>Bundle Download</td>
<td>File Detail</td>
<td>File Download</td>
<td>Heuristic View</td>
</tr>
<tr>
<td>Archive Manage</td>
<td>Archive Trigger</td>
<td>Archive View</td>
<td></td>
<td></td>
</tr>
<tr>
<td>On Behalf Of Access</td>
<td>Replay System</td>
<td>Replay Trigger</td>
<td>Safelist Manage</td>
<td></td>
</tr>
<tr>
<td>Safelist View</td>
<td>Manage Current User</td>
<td>Signature Download</td>
<td>Signature Import</td>
<td></td>
</tr>
<tr>
<td>Signature Manage</td>
<td>Signature View</td>
<td>Submission Create</td>
<td>Submission Delete</td>
<td></td>
</tr>
<tr>
<td>Submission Manage</td>
<td>Submission View</td>
<td>Workflow Manage</td>
<td>Workflow View</td>
<td></td>
</tr>
</tbody>
</table>

- Our APIs also perform data management/administration automatically
So... WHAT can we use to communicate with the API?

- Common methods of API interaction are, but not limited to:
  - Assemblyline Client (Python/Java)
  - Assemblyline Client from the CMD (Python/Java)
  - CURL
  - HTTP library in any programming language
So... HOW do we use the API?

- The API is fully documented and available on your instance at: /help/api

- Extended documentation available at: https://cybercentrecanada.github.io/assemblyline4_docs/integration/ingestion_method/
Assemblyline Client

• The API client is available in Python and Java
  • https://pypi.org/project/assemblyline-client/
  • https://github.com/CybercentreCanada/assemblyline-java-client

Here is how to initialize it in Python:

```python
from assemblyline_client import get_client

# Connect/Authenticate with Assemblyline deployment
PORT = '443'
HOST = "localhost"
client = get_client(f'https://{HOST}:{PORT}', auth=('admin', 'admin'), verify=False)
```
Searching for Data

• **Search API:**
  - Search for data that might belong to certain indices/buckets (files, results, signatures) with optional filtering criteria

```plaintext
GET/POST /api/v6/search/index/

Fully documented: ✔️
Requires login: ✔️
Required user roles:
- Admin View
- Defined View
- Signature View
- Operation View
- Workflow View

API Path: /api/v6/search/index/

Description:
Search through specified index for a given query. Uses Lucene search syntax for query.

Variables:
- index: Bucket to search in (alert, submission,...)

Arguments:
- query: Query to search for

Optional Arguments:
- deep_paging_id: ID of the next page or "" to start deep paging
- filters: List of additional filter queries limit the data
- offset: Offset in the results
- rows: Number of results per page
- sort: How to sort the results (not available in deep paging)
- fl: List of fields to return
- timeout: Maximum execution time (ms)
- use_archive: Allow access to the malware archive (Default: False)
- archive_only: Only access the Malware archive (Default: False)
```
Submission Full & Summary Report (without Ontology)

- Full Submission Results
- Submission Summary
Submission report (with Ontology)

For machine-to-machine parsing, we recommend the use of the Ontology APIs

<table>
<thead>
<tr>
<th>Method</th>
<th>Endpoint Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/api/v4/ontology/alert/&lt;alert_id&gt;/</td>
<td>Get Ontology For Alert</td>
</tr>
<tr>
<td>GET</td>
<td>/api/v4/ontology/file/&lt;sha256&gt;/</td>
<td>Get Ontology For File</td>
</tr>
<tr>
<td>GET</td>
<td>/api/v4/ontology/submission/&lt;sid&gt;/</td>
<td>Get Ontology For Submission</td>
</tr>
</tbody>
</table>

See further documentation:
https://cybercentrecanada.github.io/assemblyline4_docs/odm/models/ontology/ontology/
# Submission methods

<table>
<thead>
<tr>
<th>Asynchronous (/api/v4/ingest/)</th>
<th>Synchronous (/api/v4/submit/)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Supports <strong>large volumes</strong> of files for processing</td>
<td>• Instant scanning (given highest priority to skip the queue)</td>
</tr>
<tr>
<td>• Not subjected to <strong>quota limits</strong></td>
<td>• Analysis guaranteed (no data <strong>sampling</strong>)</td>
</tr>
<tr>
<td>• <strong>Alerting</strong> functionality is used</td>
<td>• <strong>Metadata</strong> searchable for all submissions</td>
</tr>
<tr>
<td>• Performance optimizations with <strong>submission-level caching</strong></td>
<td>• <strong>Not suitable for large volumes</strong> of files</td>
</tr>
<tr>
<td>• Ingestions may be <strong>queued</strong> for an extended time or <strong>sampled</strong> based on system busyness</td>
<td>• Subjected to <strong>quota limits</strong> depending on user (Default: 5 concurrent submissions)</td>
</tr>
<tr>
<td>• <strong>Metadata</strong> associated to ingestions aren’t indexed because there is no submission entry created</td>
<td>• <strong>Alerting</strong> not available</td>
</tr>
<tr>
<td></td>
<td>• <strong>No submission-level caching</strong></td>
</tr>
</tbody>
</table>
And more...

- Search it!
Coding time!
Important Information RE: Rest API

• Use the API key, otherwise library needs to handle session cookies and XSRF tokens

• Most Assemblyline APIs are expecting to receive and return JSON**
  • Accept header "application/json"
  • Content-type header "application/json"

• All Assemblyline APIs end with trailing forward slash "/"

• Headers to authenticate are “X-USER” and “X-APIKEY"

** There are also other APIs where multipart/form-data is used (ie. Ingest)
Exercise 1: Collecting Network IOCs

Scenario:

“I want to collect all the network-related IOCs that Assemblyline was able to extract and store them in a dictionary/mapping.

For my use-case, I would also want to sort them based on the type of network IOC (ie. domain, IP, URL)”

Expected Result:

```json
{
    "network.static.ip": ["172.0.0.1", ...]
    "network.static.domain": ["www.google.com", ...]
    ...
}
```
Exercise 1: Pull Network IOCs from submission

Web APIs Involved:
GET /api/v4/submission/summary/<sid>/
GET /api/v4/ontology/submission/<sid>/

Python APIs Involved:
Client.submission.summary(<sid>)
Client.ontology.submission(<sid>)

SID: 1nAXRc365frBiSXKg0qX0Q
Exercise 1: Pull Network IOCs from submission

- Option 1 (using Submission API)

  Option 1: Get IOCs for the submission summary API
  ```python
  # client.submission.summary --> /api/v4/submission/summary/sid/
  for tag_name, tag_values in client.submission.summary(SID)['tags']['ioc'].items():
    for tag_value, tag_verdict, is_tag_safelisted, classification in tag_values:
      if tag_name.startswith('network'):
        # Create the tag category if does not exist
        COLLECTED_IOCS.setdefault(tag_name, [])
        # Add the IOC to our list of collected IOCs
        COLLECTED_IOCS[tag_name].append(tag_value)
  ```

- Option 2 (using Ontology API)

  Option 2: Get IOCs from the ontology API
  ```python
  # client.ontology.submission --> /api/v4/ontology/submission/sid/
  for record in client.ontology.submission(SID):
    for tag_name, tag_values in record['results']['tags'].items():
      if tag_name.startswith('network'):
        # Create the tag category if does not exist
        COLLECTED_IOCS.setdefault(tag_name, [])
        # Add the IOC to our list of collected IOCs
        COLLECTED_IOCS[tag_name].extend(tag_values)
  ```
**Exercise 1:** Client vs Native Requests

### Assemblyline Client

```python
# Option 1: Get IOCs for the submission summary API
# client.submission.summary --> /api/v4/submission/summary/sid/
for tag_name, tag_values in client.submission.summary[SID]['tags']['ioc'].items():
    for tag_value, tag_verdict, is_tag_safelisted, classification in tag_values:
        # Check if verdict is indeed malicious
        if tag_name.startswith('network'):
            # Create the tag category if does not exist
            COLLECTED IOCS.setdefault(tag_name, [])
            # Add the IOC to our list of collected IOCs
            COLLECTED IOCS[tag_name].append(tag_value)
```

### Python Requests

```python
data = requests.get(f'{host}/api/v4/submission/summary/{SID}/', headers=headers, verify=False).content
summary = json.loads(data)['api_response']
for tag_name, tag_values in summary['tags']['ioc'].items():
    for tag_value, tag_verdict, is_tag_safelisted, classification in tag_values:
        # Check if verdict is indeed malicious
        if tag_name.startswith('network'):
            # Create the tag category if does not exist
            COLLECTED IOCS.setdefault(tag_name, [])
            # Add the IOC to our list of collected IOCs
            COLLECTED IOCS[tag_name].append(tag_value)
```
Exercise 2: Performing Filtered File Collection

Scenario:

“I want to collect all files with a very high score in Assemblyline (score ≥ 7000).

I would like to also store these files on my AV-protected host so I can feed it to another process.”
Exercise 2: Download file(s) with a certain score

APIs Involved:
GET /api/v4/search/<index>/
GET /api/v4/submission/full/<sid>/
GET /api/v4/file/download/<sha256>/

Python APIs Involved:
Client.search.stream.<index>()
Client.submission.full(<sid>)
Client.file.download(<sha256>)
Exercise 2: Download file(s) with a certain score

```python
# For all submissions that are over the file score threshold
# client.search.stream.submission --> /api/v4/search/submission/?deep_paging_id=*  
for record in client.search.stream.submission(query=f"max_score:>{FILE_SCORE_THRESHOLD}" , fl='sid'):
    sid = record['sid']

# Download the full submission result and compute the score for each file
# client.submission.full --> /api/v4/submission/full/sid/
submission_results = client.submission.full(sid)

# Compute the score of each files in the submission
files_scores = dict()
for result in submission_results['results'].values():
    # Initialize the default score for the file if the file is not in the list
    files_scores.setdefault(result['sha256'], 0)

    # Add the score of the result record to the file
    files_scores[result['sha256']] += result['result'] ['score']

# For each files where the score is greater than threshold, download in cARTed format
# client.file.download --> /api/v4/file/download/sha256?encoding=cart/
for sha256, score in files_scores.iteritems():
    if score >= FILE_SCORE_THRESHOLD:
        client.file.download(sha256, encoding="cart", output=os.path.join(OUTPUT_DIRECTORY, f"{sha256}.cart"))
```
Exercise 3: Ingest files through Ingest API

Scenario:

“I want to be able to automate ingestion from a host-based sensor to submit files to Assemblyline and send the parsed results to a database for long-term use.”

• What are notification queues, and should I use them?
Exercise 3: continued

Web APIs Involved:
POST /api/v4/ingest/
GET /api/v4/ingest/get_message_list/<notification_queue>/

Python APIs Involved:
Client.ingest()
Client.ingest.get_message_list(<notification_queue>)
# Exercise 3: Solution

## # SENDER
# Ingest all files to scan in Assemblyline
```python
for file_path in files_to_scan:
    # That's it, just need to send all files in... the receiver will pull the results
    client.ingest(path=file_path, metadata={'file_path': file_path}, nq=NOTIFICATION_QUEUE_NAME)
```

## # RECEIVER
# Receive completion messages from the notification queue
```python
# client.ingest.get_message_list --> /api/v4/ingest/get_message_list/<NOTIFICATION_QUEUE_NAME>/
while len(files_to_scan) != 0:
    for result in client.ingest.get_message_list(NOTIFICATION_QUEUE_NAME):
        # This is the file we are receiving result for
        current_file = result['submission']['metadata']['file_path']

        # For each completion message, pull the result record to get the score
        submission = client.submission(result['submission']['sid'])

        # Print file score to screen
        print(current_file, "=", submission['max_score'])

        # Stop waiting for the file
        files_to_scan.remove(current_file)
```

# Otherwise wait for more messages until we're finished
```python
sleep(1)
```
Exercise 4: Alert monitoring and identify IOC for blocking

Let's say we want to action on IOCs that Assemblyline has alerted on

**Web APIs Involved:**
GET /api/v4/search/<index>/

**Python APIs Involved:**
Client.search.<index>

```python
# Search through the alert index for alerts with IPs and Domains
# client.search.alert --> /api/v4/search/index/
for alert in client.search.alert('al.ip:* OR al.domain:* OR al.uri:*', f1="al.detailed.*")['items']:
    # Iterate over the IOCs in the alert
    for ioc_type in ['ip', 'domain', 'uri']:
        # Iterate through the different items to check if they should be blocked
        for ioc in alert['al']['detailed'][ioc_type]:
            # Make sure those IOCs are not safe or informational
            if ioc['verdict'] in ['info', 'safe']:
                continue
            # Block suspicious and malicious IOCs (ie. add to FW rules)
            block_IOC(ioc=ioc['value'], ioc_type=ioc_type, verdict=ioc['verdict'])
```
**Exercise 5:** What about custom tradecraft?

Scenario:

“I can't use Assemblyline's Python/Java client to integrate with my existing tradecraft. What can I do?”

- Can I use cURL, Postman, or any other compiled application?
Exercise 5: CURL

• Submit a file using the "Submit" transmission method using Raw HTTP/Curl

  client.submit \rightarrow /api/v4/submit/

• Ingest a file using the "Ingest" transmission method using Raw HTTP/Curl

  client.ingest \rightarrow /api/v4/ingest/
How are services built?

The different parts that compose a service
Creating new services

• Bare minimum:
  • Python file with a ServiceBase class that implements the execute function
  • service_manifest.yml
  • Dockerfile*

• Service Manifest:
  • name, version, description
  • accepts, rejects (file types that you are interested into)
  • file_required, timeout, stage, category
  • config, submission_params
  • heuristics
  • docker_config, dependencies, update_config
ServiceBase class

- Overwritable functions
  - __init__()
  - _load_rules()
  - start()
  - execute(request: ServiceRequest)

- self.config
  - service_manifest.yaml: config

- self.working_directory

- self.log.(debug|info|warning|error)
Execution – ServiceRequest object

- request.file_type, file_path, file_contents
- request.get_param()
  - service_manifest.yaml: submission_params
- request.add_extracted()
- request.add_supplementary()

- ResultSection
  - request.result = Result()
Result & ResultSections

• A Result contains ResultSections
• What can a ResultSection contain:
  • Body of information, with associated format
  • Classification
  • Tags
  • One Heuristic
    • Score
    • Signatures
      • More score

☐ The score of the heuristic is applied to all content of the ResultSection
ResultSection - Text

Example of a default section

- transition technologies
- programs marketplace To complex private website Government support
- BCIP Program academia in services Canada collaborating working market experts new invite services

- [HEURISTIC] Extraction config information
- [SIGNATURE] sig_one
- [SIGNATURE] sig_two
- [SIGNATURE] sig_three
- [SIGNATURE] sig_four
- [ACTOR] MUSTANG PANDA
- [IMPLANT] RESULTSAMPLE
- [IMPLANT] ASTAROTH
- [ATT&CK] Shared Modules
- [ATT&CK] Clipboard Data
- [ATT&CK] Regsvr32
- [ATT&CK] JavaScript
- [ATT&CK] NTFS File Attributes
- [ATT&CK] Registry Run Keys / Startup Folder
- [ATT&CK] Dead Drop Resolver
- [ATT&CK] Obfuscated Files or Information
ResultSection - KeyValue

- Can be sorted or ordered

TLP:C :: Example of a KEY_VALUE section

| A Bool | false |
| A Str  | Some string |
| An Int | 102 |
| Key    | value |

TLP:C :: Example of an ORDERED_KEY_VALUE section

| Key0   | value0 |
| Key1   | value1 |
| Key2   | value2 |
| Key3   | value3 |
| Key4   | value4 |
ResultSection - Table

- Can be nested to a maximum of two deep

<table>
<thead>
<tr>
<th>A Bool</th>
<th>A Str</th>
<th>An Int</th>
<th>Extra Column Here</th>
<th>Extra Column There</th>
<th>Nested Key Value Pair</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>Some string1</td>
<td>101</td>
<td>confirmed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>true</td>
<td>Some string2</td>
<td>102</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>false</td>
<td>Some string3</td>
<td>103</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Some string4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Some string4: -10000000000
0000000000

confirmed

false
A Str
Nested Kv That's Too Deep
{"a_bool":false,"a_str":"Some string3","an_int":103}
ResultSection - Image
Example of a JSON section

```json
{
  6 items
  "a_bool" : false
  "a_dict" : { 2 items
    "bool" : true
    "list_of_dict" : [ 2 items
      0 : { 2 items
        "d1_key" : "val"
        "d1_key2" : "val2"
      }
      1 : { 2 items
        "d2_key" : "val"
        "d2_key2" : "val2"
      }
    ]
  }
  "a_list" : [ 3 items
    0 : "a"
    1 : "b"
    2 : "c"
  ]
}
```
ResultSection - MemoryDump

Example of a memory dump section

HEURISTIC] Config decoding

This is some random text that we will format as an hexdump and you'll see that the hexdump formatting will be preserved by the memory dump section!
ResultSection - Graph

Example of colormap result section:

- Value 0
- Value 20
ResultSection - Process Tree

- Coloured based on the score of the process-associated signatures
ResultSection - Timeline

TLP:C :: Timeline

- Value: 0
  - NODE 0
  - Description: 0
- Value: 1
  - NODE 1
  - Description: 1
- Value: 2
  - NODE 2
  - Description: 2
- Value: 3
  - NODE 3
  - Description: 3
ResultSection - MultiSection

Example of Multi-typed section

We have detected very high entropy multiple sections of your file, this section is most-likely packed or encrypted.

Here are affected sections:

<table>
<thead>
<tr>
<th>Offset</th>
<th>0x008000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section Name</td>
<td>.UPX0</td>
</tr>
<tr>
<td>Size</td>
<td>4196 bytes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Offset</th>
<th>0x009000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section Name</td>
<td>.UPX1</td>
</tr>
<tr>
<td>Size</td>
<td>4196 bytes</td>
</tr>
</tbody>
</table>
How to run it

- python -m assemblyline_v4_service.dev.run_service_once <your_service> <sample>

- Important for full deployment
  - The service_manifest.yml's version needs to fit your deployment's
Good examples – ElfParser

• Package a compiled executable
• Parse the output of the executable to fill ResultSections for the user

https://github.com/CybercentreCanada/assemblyline-service-elfparser
Good examples – Api Vector

- Use a public library (apiscout, lief)
- Load an external file
- Use an updater

https://github.com/CybercentreCanada/assemblyline-service-apivector
Good examples – UrlDownloader

• stage: POST
• file_required: false
• is_external, allow_internet_access: true
• uses_tag_scores, uses_metadata, uses_temp_submission_data: true

https://github.com/CybercentreCanada/assemblyline-service-urlDownloader/
Workshop time!
Creating the new MBInfo module

- **Workshop:**
  https://github.com/CybercentreCanada/assemblyline-training-first2023

- **Documentation:**
  https://cybercentrecanada.github.io/assemblyline4_docs/

- **MalwareBazaar:**
  https://bazaar.abuse.ch/
Time to get serious
How to get your deployment ready for multiple millions of files
Preface

Based on our current biggest production environment
Node

- Don’t use nodes that are too small, Elastic/Redis can use a lot of resources
  - Minimum: 8 cores / 32 GB
  - What we use: 16 cores / 64 GB
- The minimum amount of nodes required by your cluster is the amount of Elastic pods that you have
  - We have 12 Elastic pods so our deployment auto-scales from 12 nodes to 72 nodes
Ingestion

• For high volume ingestion, **do not use** /api/v4/submit/
• Use this instead: /api/v4/ingest/
  • Tailored for rate limiting if AL can't keep up
  • Will queue submission for processing later
• If ingestion slows down the UI because the rate is too high
  • `separateIngestAPI: true` in your `values.yml` files
  • Spins up dedicated pods for ingestion
File storage

• Do not use the provided minio container for file storage
  • Not that minio is not good, we just haven’t spent any effort making the chart deploy it correctly
• Use either:
  • Azure blob storage, if you are on AKS
  • Amazon S3 if you are on AWS
  • Deploy your own Minio with redundancy or any other well-supported S3 compatible file storage
• Don’t put your file storage secrets in your values.yml file, use Kubernetes secrets instead
• Example:

```yaml
internalFilestore: false
configuration:
  filestore:
    storage:
      - "azure://<blob_store_name>.blob.core.windows.net/storage?access_key=${FILESTORE_PASSWORD}"
    cache:
      - "azure://<blob_store_name>.blob.core.windows.net/cache?access_key=${FILESTORE_PASSWORD}"```
Redis

- All messaging passed to services and Dispatcher/Ingester-shared memory space is stored in Redis
- Redis is our only component that cannot be scaled
- You should tweak Ram / CPU / Threads requirements to fit your need
  - We use the following values in values.yml:
    
    ```yaml
    redisVolatileIOThreads: 5
    redisVolatileReqCPU: 4
    redisVolatileLimCPU: 4
    redisVolatileReqRam: 4Gi
    
    redisPersistentIOThreads: 3
    redisPersistentReqCPU: 2
    redisPersistentLimCPU: 2
    redisPersistentReqRam: 8Gi
    redisPersistentLimRam: 32Gi
    ```
Dispatcher

• You can change the number of threads Dispatcher uses
• Also make sure Dispatcher is reserved a full core and has enough RAM
  • *NOTE: It’s a Python process so don’t give it more than a core*
• We use the following `values.yml` config:

```yaml
disptacherShutdownGrace: 1800
dispatcherResultThreads: 8
dispatcherFinalizeThreads: 8
dispatcherReqCPU: 1
dispatcherLimCPU: 1
dispatcherReqRam: 2Gi
dispatcherLimRam: 4Gi
```
Expiry

- With big data input comes big data deletion
- We gave Expiry more cores and more workers to be able to expire all that data
  - Here what we use in our `values.yml`:
    ```yaml
    expiryReqCPU: 2
    expiryLimCPU: 4
    configuration:
      core:
        expiry:
          workers: 50
          delete_workers: 5
    ```
Scaling

- Use **cpu_overallocation** to make sure the cloud node autoscaler works
  - Use a value between 1.05 to 1.10 (105% to 110%)

- **overallocation_node_limit** will determine your maximum amount of nodes

- **min_instances** determines the minimum number of service pods loaded
  - We use 2 so our reaction time is faster but that costs more money

- **cpu_reservation** is the percentage of the required max CPU for a service that will be reserved by Kubernetes
  - The higher the value, the less time the services fight for CPU time as their CPU usage is reserved, but that comes at the price of a higher cost!

Our **values.yml** looks like this:

```yaml
configuration:
  core:
    scaler:
      cpu_overallocation: 1.05
      overallocation_node_limit: 72
    service_defaults:
      min_instances: 2
  services:
    cpu_reservation: 0.7
```
Auto-scalers

• The scaler component is dedicated to managing services
• To make sure you have enough core components to handle the service load you can adjust the max number of components in the `values.yml` files
  • Here’s how we’ve setup ours:

    ```yaml
    dispatcherInstancesMax: 25
    ingestAPI InstancesMax: 50
    serviceServerInstancesMax: 50
    dispatcherTargetUsage: 40
    ```
Datastore

- Because you’ll have more data you’ll need more Elastic pods
- To make the most out of those pods they will need more CPU
  - Match the request / limit of CPU so Elastic does not fight with services for CPU time.
- The size of the index will be larger, Elastic will need more RAM to process the queries
- To take advantage of the distributed computing, since Elastic has more nodes, it will need more shards so each node gets busy enough
  - If you’ve deployed your cluster before adjusting the shard, you’ll have to use the `fix_shards` CLI command to edit the shard count on affected indices

Our biggest production system has 4.7TB of index with 1.8 Billion documents

Our `values.yml` looks like this:

```
elasticEmptyResultShards: 16
elasticFileShards: 16
elasticResultShards: 36
elasticSubmissionShards: 24
datastore:
  replicas: 12
  resources:
    requests:
      cpu: 4
      memory: 12Gi
    limits:
      cpu: 4
      memory: 20Gi
```
What does the future hold for Assemblyline?
Malware Archive

- Save Assemblyline submissions forever
- More file-centric view of Assemblyline with the ability to:
  - Add comments on files
  - Add labels to files
  - Find related files based on tags/labels
  - See trends for different tags/labels
- The file/submission part of the malware archive will be able to be searched/browsed as part as the live data as well
Yara Retro-hunt

- Run a Yara rule on the full file set of Assemblyline or on files kept in the archive only
- View the progress of your scan
- View previous Retro-hunt scans by you or any other users in the system
- Supports the classification engine so you can limit who can see the scan and the files that are returned from the hunt are only files that you can see
External query plugins

- Allow the Assemblyline API/UI to query external sources for hashes and IOCs using a plugin interface
- Plugins are:
  - Micro relay web services that you load in your infrastructure
  - Have a defined output that the UI can display
  - Only a small configuration is needed so the UI knows the plugin exists
  - Template and examples will be available so you can have inspiration to write your own for your own services
- Plugins that will be available out-of-the-box:
  - VirusTotal
  - Malware Bazaar
  - Another AL instance
That's all folks!

Get in touch with us if you need help or want to build a closer relationship with our team

assemblyline@cyber.gc.ca
discord.gg/GUAy9wErNu