CANADIAN CENTRE FOR CYBER SECURITY

Supercharge your Malware Analysis Workflow with Assemblyline

The A(ssemblyline)-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?

20 min (9:30 – 9:50)

- User Interface Showcase

1h (9:50 – 10:50)

Coffee Break

- Assemblyline API Walkthrough

1h 20min (11:20 – 12:40)

Lunch

- The different parts of a service

20min (14:00 – 14:20)

- Service Creation

1h (14:20 – 15:20)

Coffee Break

- Service Creation (Wrap up)

1h (15:50 – 16:50)

- Scale your deployment
- Future Work

20min (16:50 – 17:10)

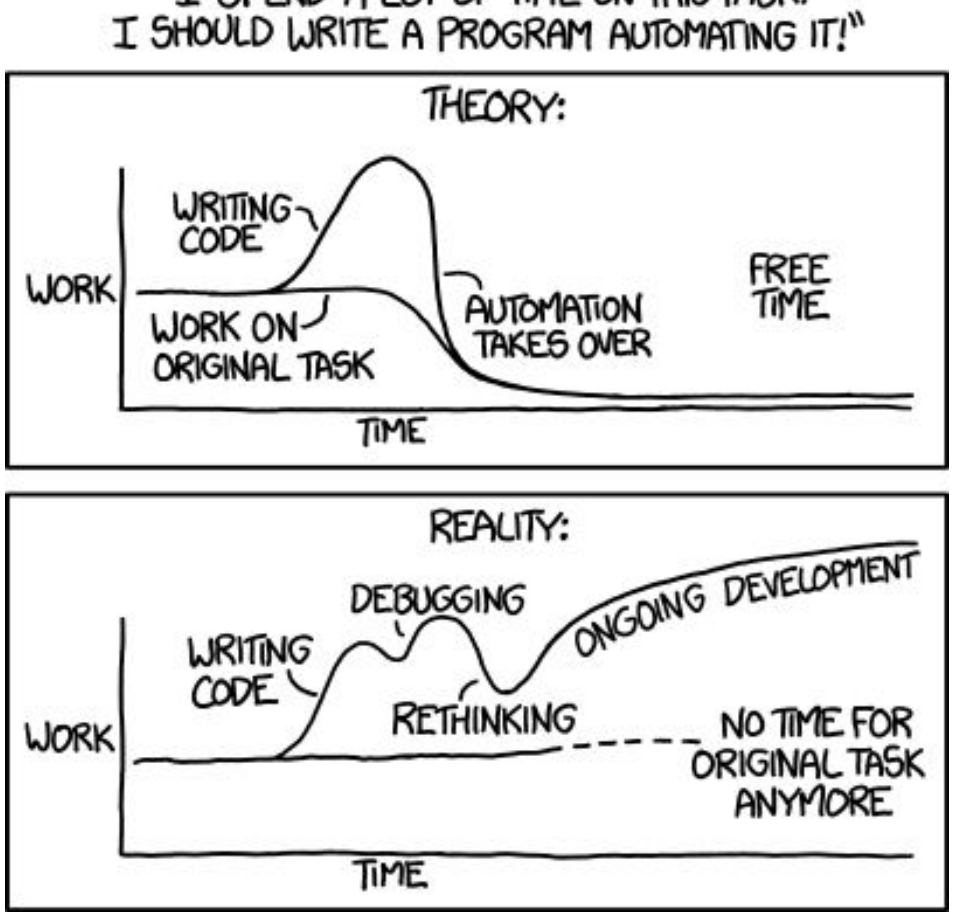
A Little Bit of History

… Back in the days

Small team of 3 Reverse Engineers

Daily Stats:

- ~ 10 files received
- ~ 5 Unique
- ~ Between 0 – 5 analysed



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

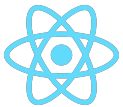
- Distributed analysis platform
- Aggressive deduplication
- Alerting system with automated workflows
- Scalable
- Open source since Oct 2017

- 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...

ASSEMBLYLINE

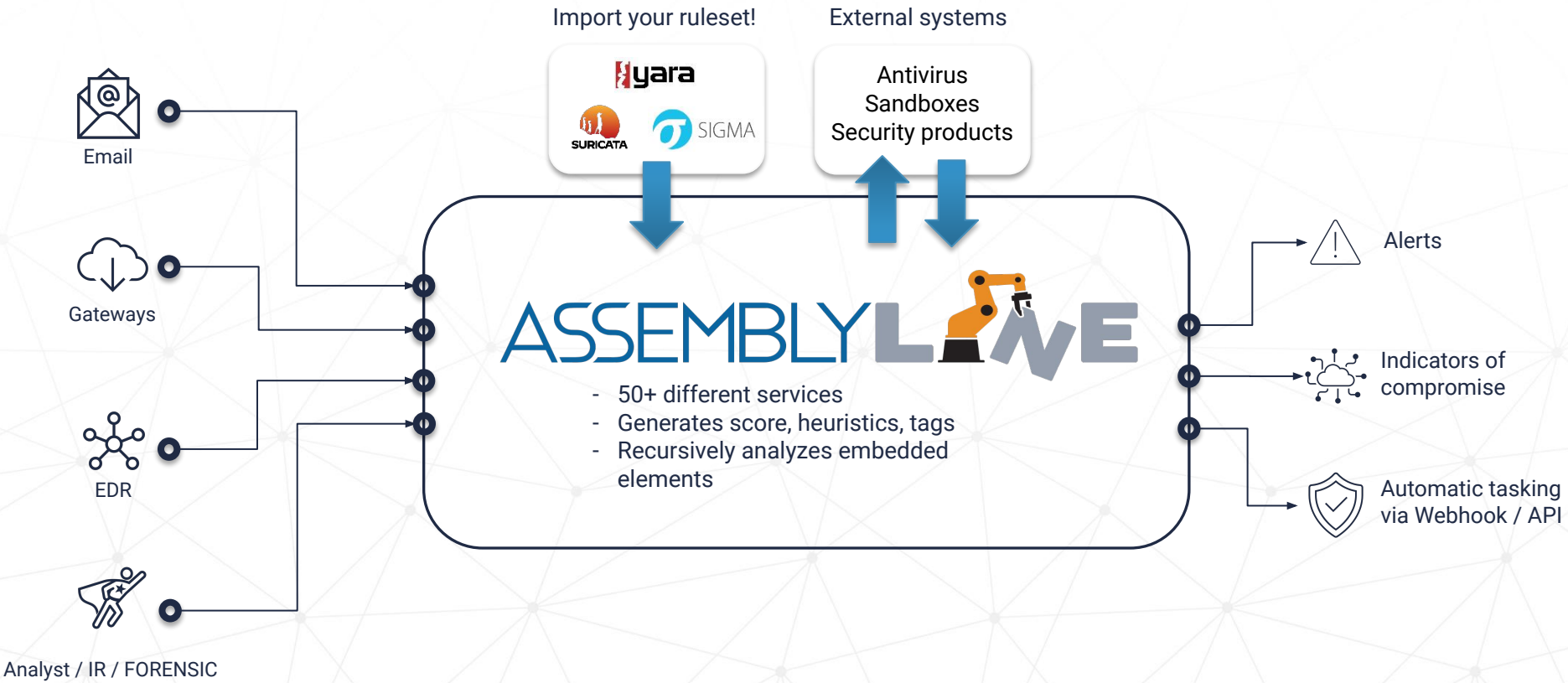


Now we're talking!

- 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?



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

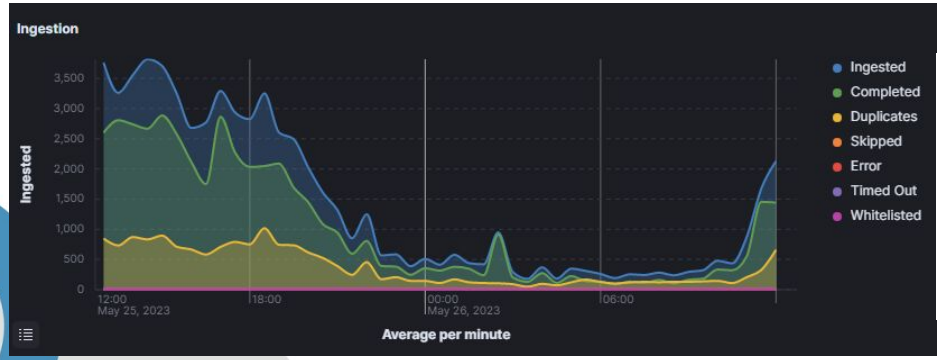
Creating your deployment

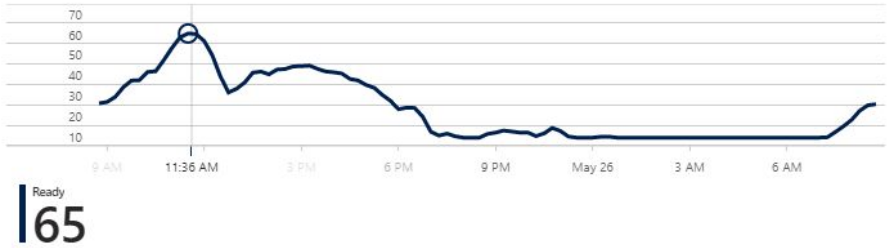
Choosing the right deployment type

- 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

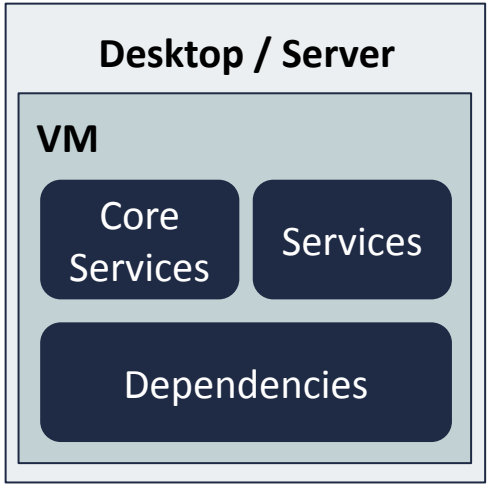
- Auto-scalable 12-72 nodes cluster (16 cores/64 GB per node)
- Up to 3.5M+ unique submissions a day (Avg. 1.5M)
- Lots of downtime during the night
- Rarely uses the full node capacity
- Mixed file types, mix of static and dynamic analysis.





Development VM using an IDE

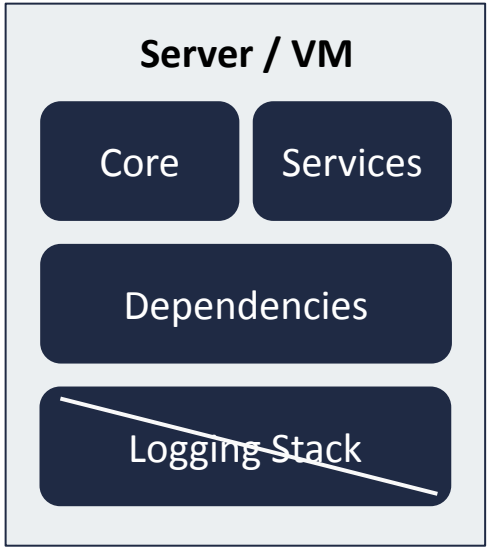
- Throughput: A couple at a time
- Only includes components that play nice with virtualization
- Easy to set up
- Wants its own operating system
  - Needs a VM, not a container
- Like the name says, this is for development, nothing else
- Minimum resources: 2 cores / 6GB of ram



* *https://cybercentrecanada.github.io/assemblyline4_docs/developer_manual/env/vscode/setup_script/*

Appliance (docker compose)

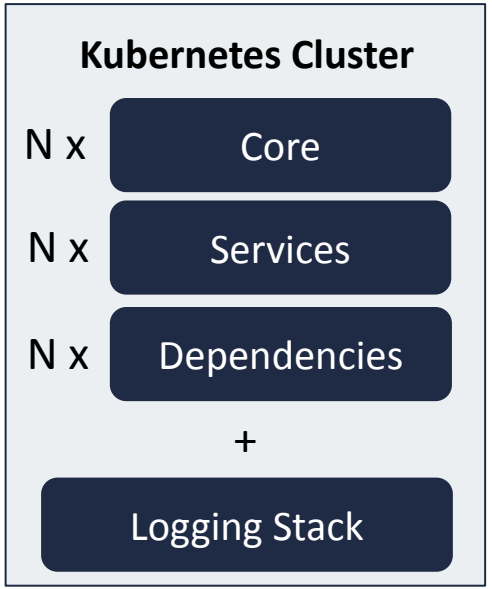
- Small throughput
- Few hundred per minute
- Fairly easy to set up
- Can be installed on a server or a VM
- Everything on the same box
- Not recommended to use with logging stack since everything is on the same box



** 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

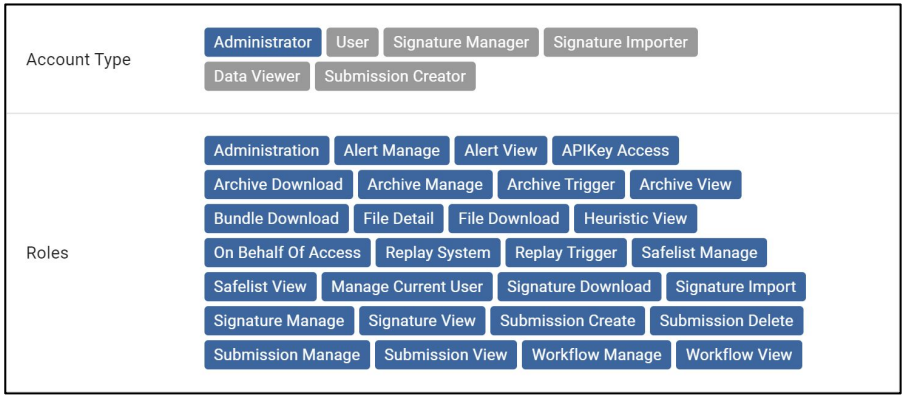
We have a test deployment ready for you…

- **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)



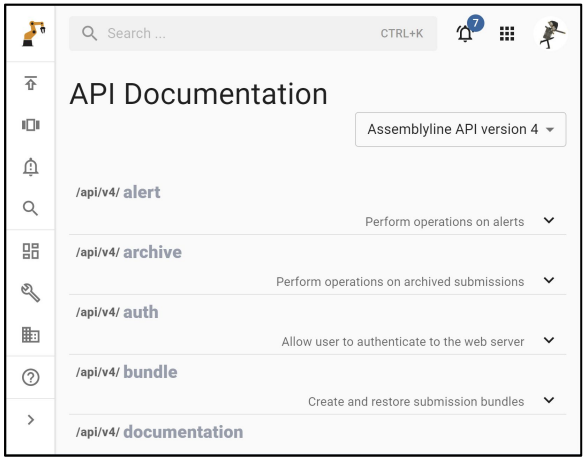
- Our APIs also perform data management/administration automatically

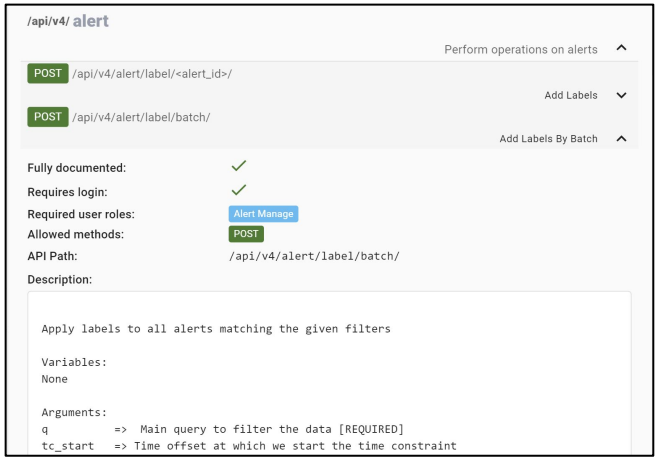
So… WHAT can we use to communicate with the API?

- 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`

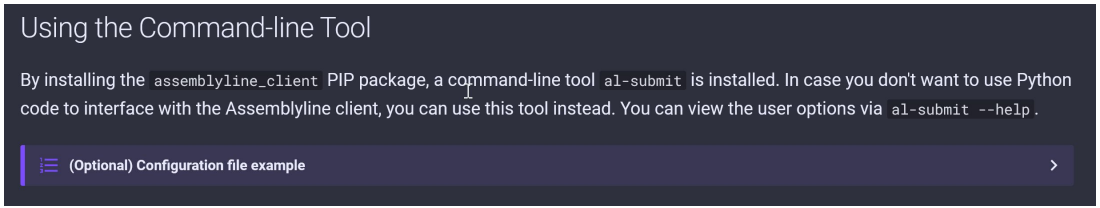




- Extended documentation available at: https://cybercentrecanada.github.io/assemblyline4_docs/integration/ingestion_method/

Assemblyline Client

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



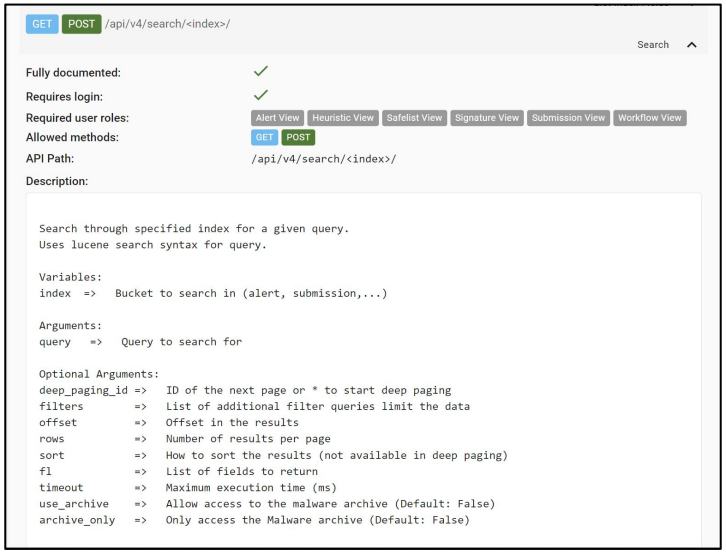
- Here is how to initialize it in Python:

```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)
```

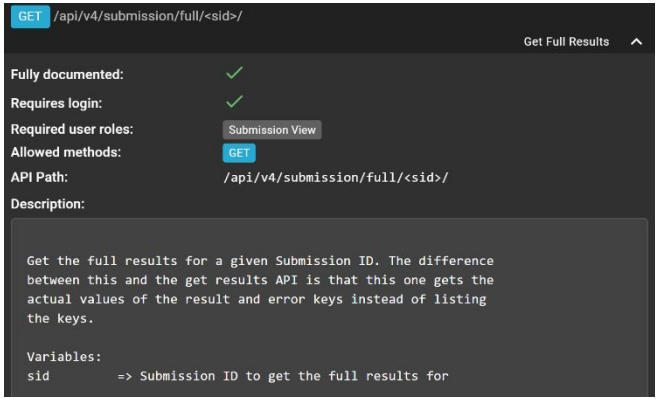
Searching for Data

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

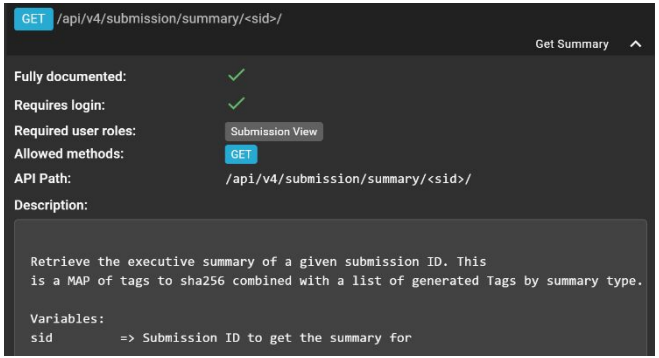


Submission Full & Summary Report (without Ontology)

- Full Submission Results



- Submission Summary



Submission report (with Ontology)

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

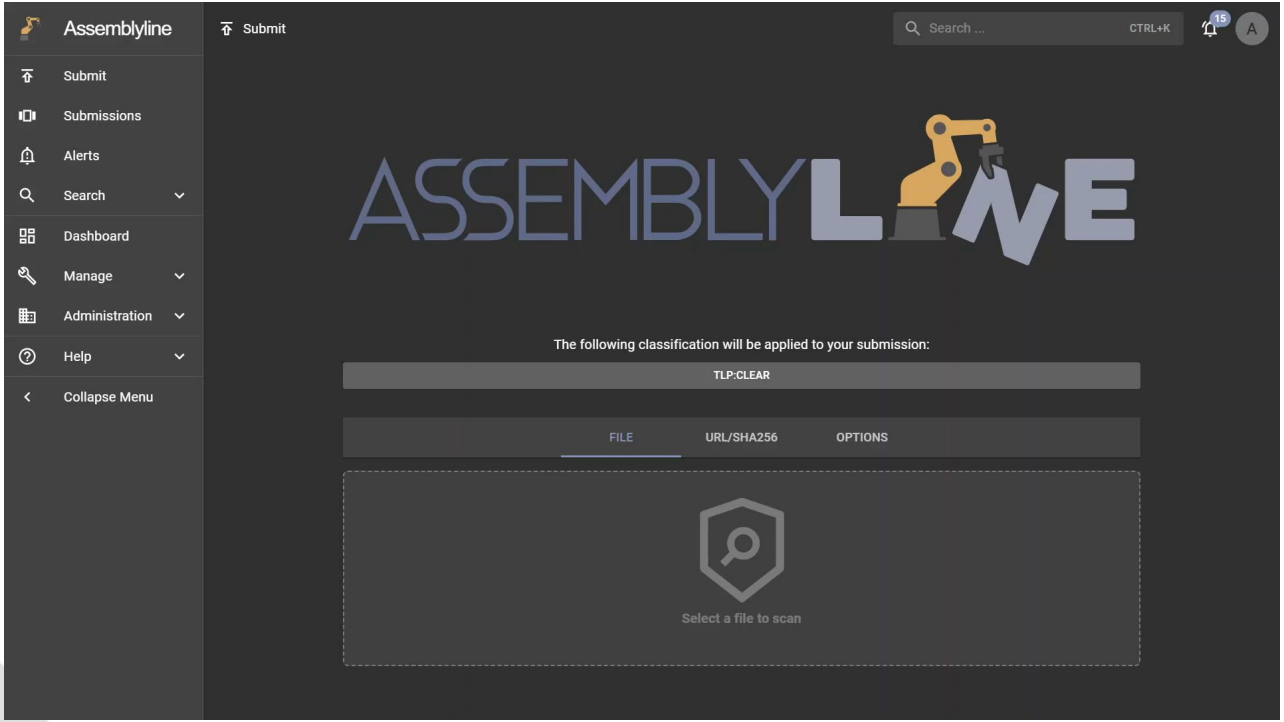
See further documentation:
https://cybercentrecanada.github.io/assemblyline4_docs/odm/models/ontology/ontology/

Submission methods

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

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:

```
{
    “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)

```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)
```

- Option 2 (using Ontology API)

```python
# Option 2: Get IOCs from the ontology API
# 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

```
# 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)
```

```
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)
```

Python Requests

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.items():
        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
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
# 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 receiveing 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
    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:*', fl="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 → /api/v4/submit/
```

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

```
client.ingest → /api/v4/ingest/
```

How are services built?

The different parts that compose a service

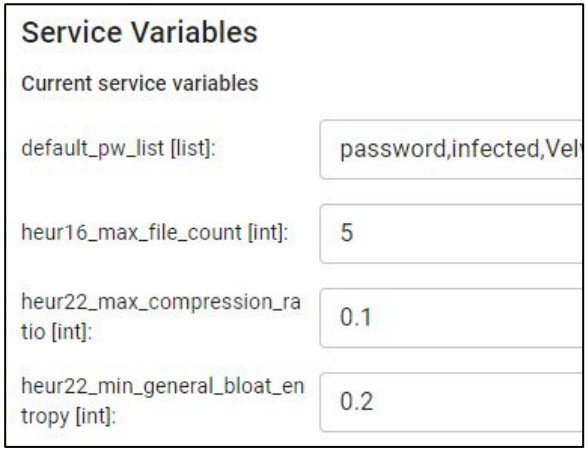
Creating new services

- 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

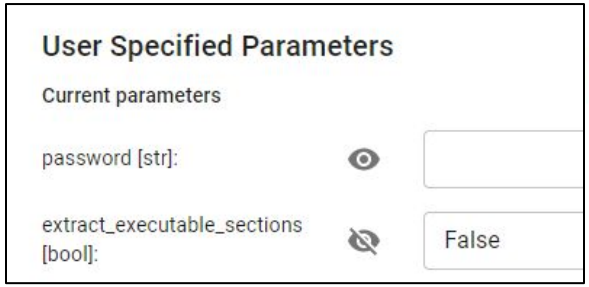
- Overwritable functions
  - __init__()
  - _load_rules()
  - start()
  - execute(request: ServiceRequest)
- self.config
  - service_manifest.yaml: config
- self.working_directory
- self.log.(debug|info|warning|error)

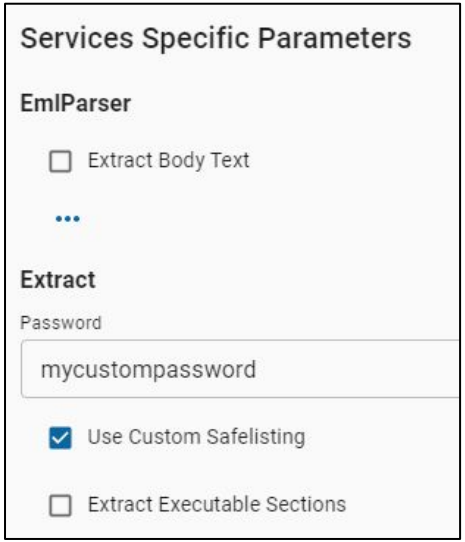


Execution – ServiceRequest object

- 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

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
```

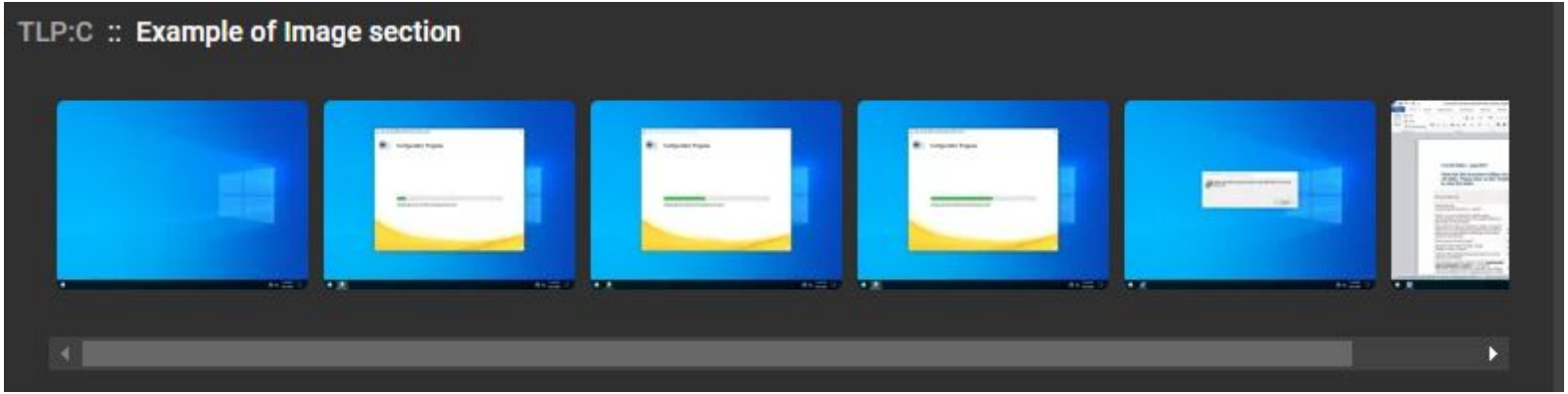
ResultSection - Table

- Can be nested to a maximum of two deep

TLP:C :: **Example of a TABLE section**

A Bool	A Str	An Int	Extra Column Here	Extra Column There	Nested Key Value Pair	
false	Some string1	101	confirmed			
true	Some string2	102				
false	Some string3	103				
	Some string4	-1000000000000000000		confirmed	A Bool A Str Nested Kv Thats Too Deep	false Some string3 {"a_bool":false,"a_str":"Some string3","an_int":103}

ResultSection - Image



ResultSection - JSON

TLP:C :: Example of a JSON section


```
{ 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"
  ]
```

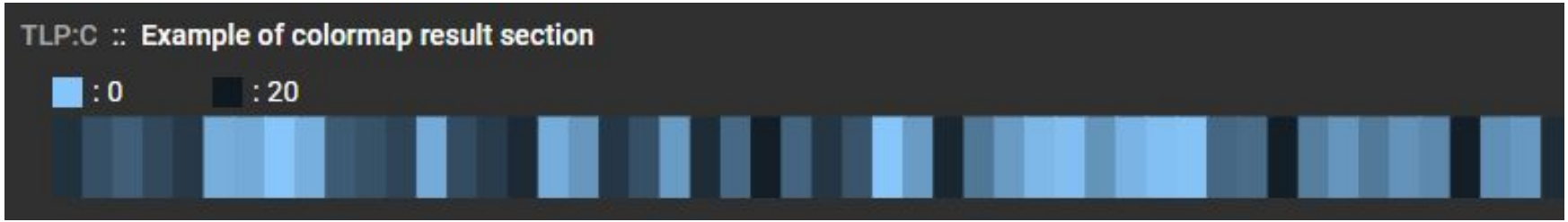
ResultSection - MemoryDump

TLP:C :: I :: Example of a memory dump section

```
00000000:  54 68 69 73 20 69 73 20 73 6f 6d 65 20 72 61 6e   This is some ran
00000010:  64 6f 6d 20 74 65 78 74 20 74 68 61 74 20 77 65   dom text that we
00000020:  20 77 69 6c 6c 20 66 6f 72 6d 61 74 20 61 73 20    will format as
00000030:  61 6e 20 68 65 78 64 75 6d 70 20 61 6e 64 20 79   an hexdump and y
00000040:  6f 75 27 6c 6c 20 73 65 65 20 74 68 61 74 20 74   ou'll see that t
00000050:  68 65 20 68 65 78 64 75 6d 70 20 66 6f 72 6d 61   he hexdump forma
00000060:  74 74 69 6e 67 20 77 69 6c 6c 20 62 65 20 70 72   tting will be pr
00000070:  65 73 65 72 76 65 64 20 62 79 20 74 68 65 20 6d   eserved by the m
00000080:  65 6d 6f 72 79 20 64 75 6d 70 20 73 65 63 74 69   emory dump secti
00000090:  6f 6e 21                                          on!
```

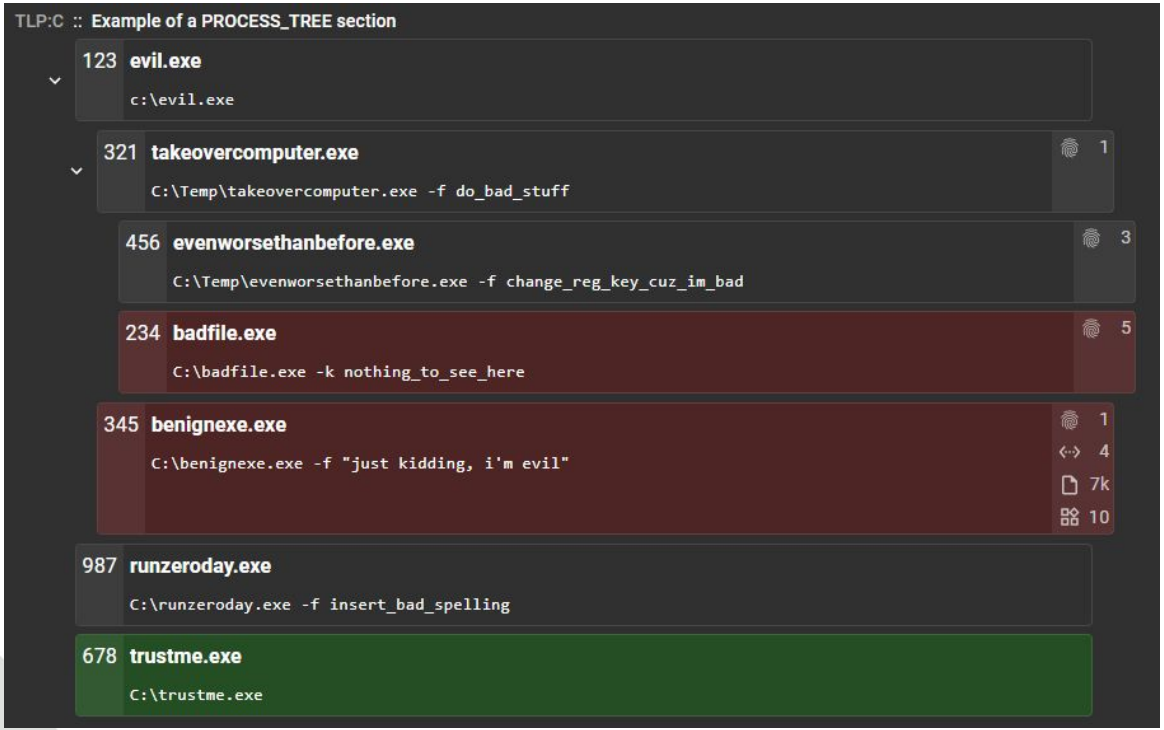
[HEURISTIC] Config decoding

ResultSection - Graph

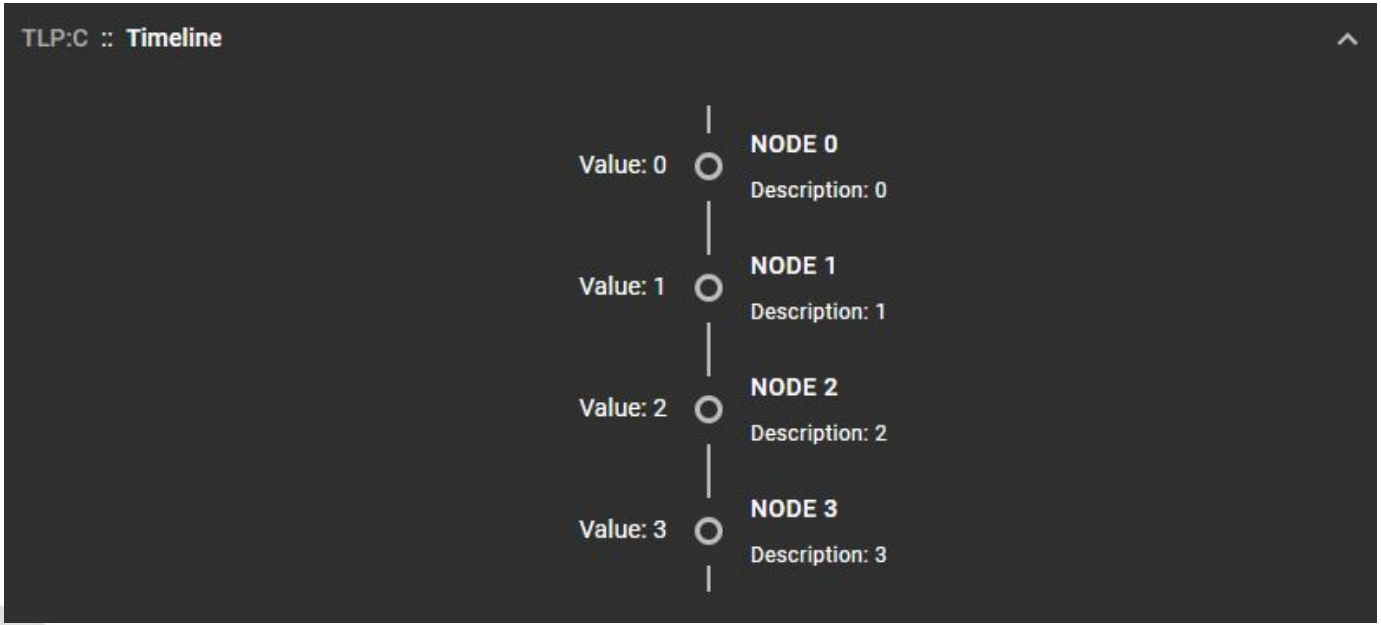


ResultSection - Process Tree

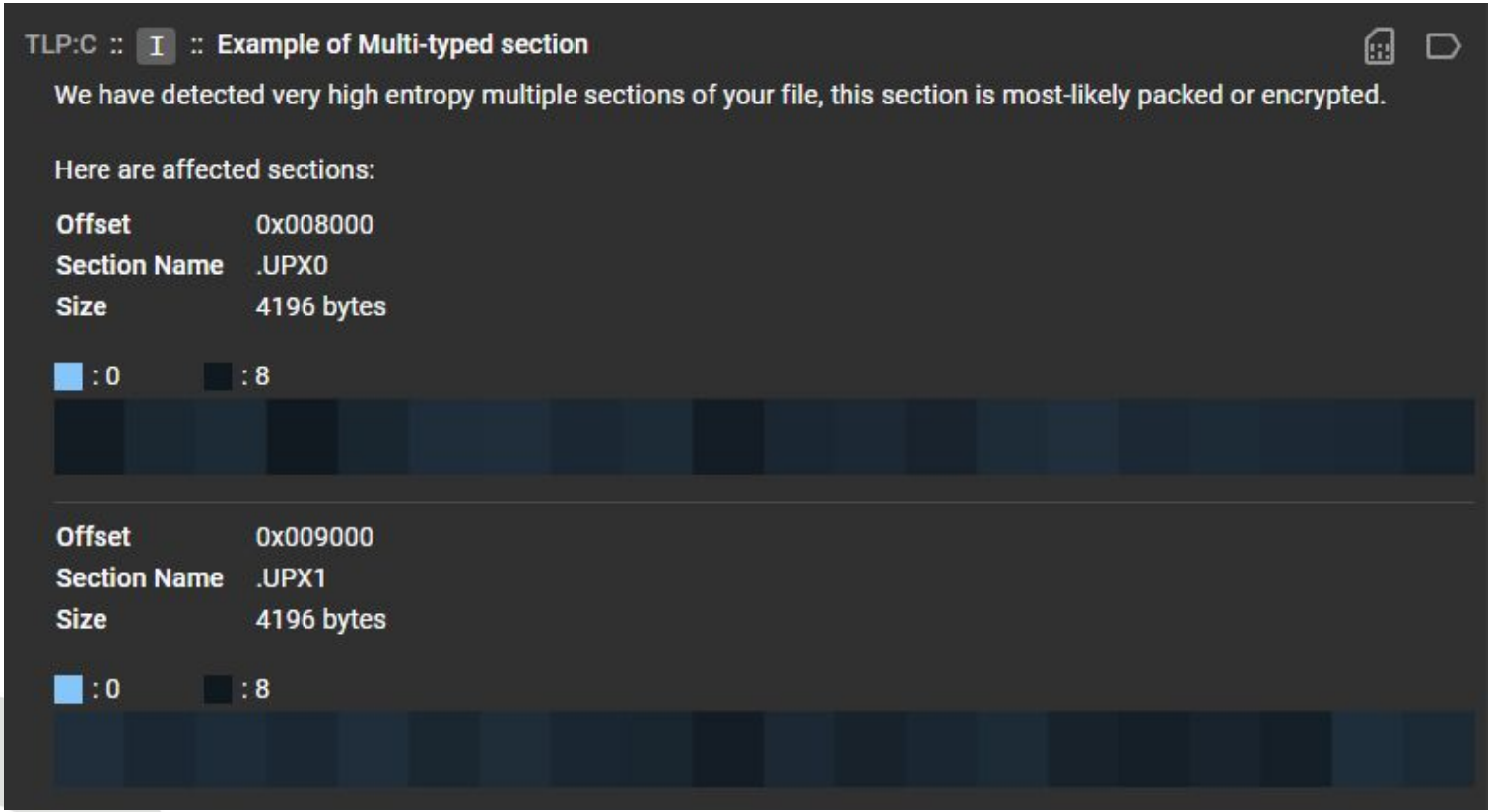
- Coloured based on the score of the process-associated signatures



ResultSection - Timeline



ResultSection - MultiSection



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

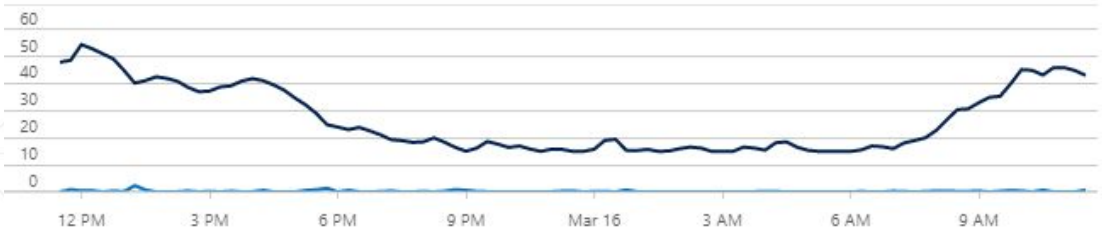
How to get your deployment ready for multiple millions of files

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:

```
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**:

```
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:

```
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**:

```
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 maxmimum 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:

```
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:

```
dispatcherInstancesMax: 25
ingestAPIInstancesMax: 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