



D7.2


**Specification of the of Privacy & Data
Protection (PrivacyNet) Orchestrator**

Project number:	833683
Project acronym:	CyberSANE
Project title:	Cyber Security Incident Handling, Warning and Response System for the European Critical Infrastructures
Start date of the project:	1st September, 2019
Duration:	36 months
Programme:	H2020-SU-ICT-2018

Deliverable type:	Report
Deliverable reference number:	DS-01-833683 / D7.2 / Final 1.0
Work package contributing to the deliverable:	WP 7
Due date:	30 June 2021
Actual submission date:	17 February 2022

Responsible organisation:	PDMFC
----------------------------------	-------

Editor:	Luís Landeiro Ribeiro
Dissemination level:	CO
Revision:	< Final 1.0 >

Abstract:	<p>This deliverable reports on the outcomes of task T7.4 Implementation of the Privacy & Data Protection (PrivacyNet) Orchestrator.</p> <p>It details the description of the implementation of the privacy services and the protection orchestrator on the CyberSANE Platform. It also contains the installation / deployment guide of the privacy services and protection orchestrator component. Work from tasks T7.2, T7.3 were relevant for defining the set of services and functionalities that the PrivacyNet makes available to the CyberSANE Platform.</p>
Keywords:	CyberSANE services, interoperability, data exchange and sharing, privacy, encryption, anonymization, API
	<p>The project CyberSANE has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 833683.</p>

Editor

Luís Landeiro Ribeiro (PDMFC)

Contributors (ordered according to beneficiary numbers)

Name	Partner	email
Jorge Martins	PDMFC	jorge.martins@pdmfc.com
Daniel Ascensão	PDMFC	daniel.ascensao@pdmfc.com
Luís Miguel Campos	PDMFC	luis.campos@pdmfc.com
Luís Landeiro Ribeiro	PDMFC	luis.ribeiro@pdmfc.com
Stylianios Karagiannis	PDMFC	Stylianios.karagiannis@pdmfc.com
Oleksii Osliak	CNR	oleksii.osliak@iit.cnr.it
Sergio Zamarripa	S2	sergio.zamarripa@s2grupo.es
Thanos Karantjias	MAG	thanos.karantjias@maggioli.gr
Sofia Karagiorgou	UBI	skaragiorgou@ubitech.eu

Version History

Version	Date	Comments, Changes, Status	Authors, Contributors, Reviewers
0.1	5/5/2021	First Draft ToC	Daniel Ascensão (PDMFC)
0.2	8/6/2021	Final TOC	Daniel Ascensão (PDMFC)
0.3	18/6/2021	First Document Draft	Oleksii Osliaik Sergio Zamarripa Thanos Karantjias Sofia Karagiorgou Daniel Ascensão
0.4	22/09/2021	Contribution on Chimera and APIs	Luís Landeiro Ribeiro (PDMFC)
0.4.1	10/10/2021	General Revision	Jorge Martins (PDMFC)
0.5	02/01/2022	Contribution to 2.1 Internal data	Daniel Ascensão (PDMFC)
0.6	10/01/2022	General Revision	Luís Campos (PDMFC)
0.7	29/01/2022	Revision of PrivacyNET & CyberSANE	Luís Landeiro Ribeiro (PDMFC)
0.7.1	30/01/2022	Typos and style revision	Stylianios Karagiannis (PDMFC)
0.8	31/01/2022	Final Revision	Luís Landeiro Ribeiro (PDMFC)
0.9	31/01/2022	Added Annex I & II	Luís Landeiro Ribeiro (PDMFC)
0.91	16/02/2022	Quality Peer Review	Haris Mouratidis & Guillermo Yuste (ATOS)
1.0	17/02/2022	Final version	Jorge Martins (PDMFC)

Disclaimer

The information in this document is provided “as is”, and no guarantee or warranty is given that the information is fit for any particular purpose. The content of this document reflects only the author’s view – the European Commission is not responsible for any use that may be made of the information it contains. The users use the information at their sole risk and liability.

Executive Summary

This deliverable reports on the specifications, configuration details and the requirements alignment of the PrivacyNet Component with the CyberSANE Platform, the other components, and the *Critical Infrastructure (CI)* owners' business needs. We present the approach we followed to connect and embed the various services and functionalities of the tools into a unified and reusable solution, which has been integrated in the CyberSANE Platform.

The PrivacyNet component provides the necessary anonymization features that allow threat intelligence and information sharing capabilities within the CIs and with relevant parties (e.g. industry cooperation groups, Computer Security Incident Response Teams - CSIRTs) in a safe way, keeping the personal identified information secret, but keeping the standard formats intact. Interoperability with 3rd party platforms is feasible, supporting standard API's such as SQL and other common protocols.

PrivacyNet delivers its features through *Application Programming Interfaces (APIs)* adopting open interoperability standards, including HTTP, JSON and STIX 2.x support.

In this deliverable, we also report how the PrivacyNet services interact with other components and the platform, especially with WP6 ShareNet services, which rely on PrivacyNet for operating the anonymization, encryption, and policy enforcement.

Contents

Executive Summary	4
Contents.....	5
Terminology, Glossary, Abbreviations	8
List of Figures	10
List of Tables	11
1 Introduction	12
1.1 Scope	12
1.2 Contribution to other Work Packages.....	12
1.3 Structure of the Document.....	13
2 PrivacyNet Functionalities.....	14
2.1 Internal Data Structures.....	15
2.1.1 Event	15
2.1.2 Metago Object Format	16
2.1.3 Source	17
2.1.4 ReadBuffer	17
2.2 PrivacyNET system architecture.....	18
2.2.1 Configuration Files	18
2.2.1.1 Config.TOML.....	19
2.2.1.2 Rules.TOML.....	19
2.2.1.3 Configuration Folders.....	20
2.2.2 Pipelines.....	21
2.2.3 Sources	22
2.2.3.1 HTTP API.....	22
2.2.3.2 Streaming Sources	23
2.2.3.3 Batches	24
2.2.4 Sinks.....	25
2.2.4.1 SQL Databases.....	25
2.2.4.2 Elastic Search	25
2.2.4.3 URLs	26

2.2.4.4	Files.....	28
2.2.5	Core.....	28
2.2.5.1	Source Engine.....	28
2.2.5.1.1	System Configurations.....	29
2.2.5.2	Rules Engine.....	31
2.2.5.3	Rule Types	33
2.2.5.3.1	Transform Rules.....	33
2.2.5.3.2	Filtering Rules	34
2.2.5.3.3	Aggregation Rules	36
2.2.5.3.4	Generating Rules	37
2.2.5.4	PII Detection Engine	38
2.2.5.4.1	Regular Expressions.....	38
2.2.5.4.2	PII Categories	39
2.2.5.4.3	PII DSL Commands	40
2.2.5.5	Policy Engine	41
2.2.5.6	Homomorphic Functions.....	42
2.2.5.7	Data Anonymization.....	44
3	PrivacyNET & CyberSANE Integration	47
3.1.1	OpenAPI Features	48
3.1.1.1	PRI-F-010.1 Encrypt Data	48
3.1.1.2	PRI-F-010.2 Decrypt data.....	49
3.1.1.3	PRI-F-020.1 Anonymization of security incident data	49
3.1.1.4	PRI-F-020.2 Anonymization of security incident reports.....	49
3.1.1.5	PRI-F-020.3 Dynamic Data Masking.....	49
3.1.1.6	PRI-F-020.4 Map & Merge Fields.....	49
3.1.1.7	PRI-F-020.5 Filter	49
3.1.1.8	PRI-F-020.6 Validation.....	49
3.1.1.9	PRI-F-030.1 Data Encryption.....	49
3.1.1.10	PRI-F-030.2 Data Decryption	49
3.1.1.11	PRI-F-030.3 Transformation	49
3.1.1.12	PRI-F-030.4 Search	50

3.1.1.13	PRI-F-040.1 PII Detection.....	50
3.1.1.14	PRI-F-040.2 PII Redaction / Privacy Rules Workflow Engine.....	50
3.1.1.15	PRI-F-040.3 Privacy Rules Operation Metrics	50
3.1.1.16	PRI-F-050.1 Data Access Management.....	50
3.1.1.17	PRI-F-050.2 Save Data Retention.....	50
3.1.1.18	PRI-F-050.3 Retrieve Data Retention.....	50
3.1.1.19	PRI-F-050.4 Register PII Data Processing.....	50
3.1.1.20	PRI-F-050.5 Retrieve PII Data Processing Details	50
3.1.1.21	PRI-F-050.6 Notify PII Data Usage	51
3.1.1.22	PRI-F-050.7 Retrieve PII Data Processing History	51
3.1.2	Custom pipeline Examples	51
3.1.2.1	Anonymization of lessons learned.....	51
3.1.2.2	Anonymization of assets with inline rules.....	57
3.1.2.3	Anonymization of incidents	58
3.1.2.4	Anonymization of Anomalies	61
4	Conclusions and Future Directions.....	64
5	References.....	67
	Annex I – Config Spec	69
	Annex II – Rules Spec.....	74

Terminology, Glossary, Abbreviations

Abbreviation	Description / Translation
API	Application Program Interface
CI	Critical Infrastructure
CRON	Unix job scheduler
CRONTAB	<p>Cron table file, that describes what jobs should be run at what schedule.</p> <p>Example:</p> <pre># _____ minute (0 - 59) # _____ hour (0 - 23) # _____ day of the month (1 - 31) # _____ month (1 - 12) # _____ day of the week (0 - 6) (Sunday to Saturday; # 7 is also Sunday on some systems) # # * * * * * <command to execute></pre>
CSIRTs	Computer Security Incident Response Teams
CyberSANE	Cyber Security Incident Handling, Warning and Response System for the European Critical Infrastructures
DSL	Domain Specific Language
RBAC	Role-based Access Control
DSA	Data Sharing Agreement
STIX	Structured Threat Information Expression
ABE	Attribute Based Encryption
SOTA	State Of The Art

FPE	Format Preserving Encryption
RUNE	Rune literals are just 32-bit integer values (however they're untyped constants, so their type can change). They represent Unicode codepoints.
DPI	Deep packet inspection
AES-FFX	Format-preserving, Feistel-based encryption
HTTP	Hypertext transport protocol
JSON	JavaScript Object Notation
REST	Representational state transfer, a set of constraints specifying how to develop internet services for distributing multi-media data
TOML	Tom's Obvious, Minimal Language

List of Figures

Figure 1 - Event Struct.....	16
Figure 2 - PrivacyNET High Level Architecture.....	18
Figure 3 - Configuration files.....	21
Figure 4 - Pipeline Generic Example.....	21
Figure 5 - Traditional Http Request API	22
Figure 6 - Integration through local storage	23
Figure 7 - ESOUT output to elastic search.....	26
Figure 8 - Rules Processing	31
Figure 9 - Rule Types	33
Figure 10 - Filter Rule	34
Figure 11 - Aggregation Rule.....	36
Figure 12 - Generating Rule	37
Figure 13 - PII Process	38
Figure 14 - Privacy Data Retention Engine	42
Figure 15 – Homosearch	44
Figure 16 - Chimera Web Studio	45
Figure 17 - GUI for rules creation	46
Figure 18 - PrivacyNET workflow inside CyberSANE	47
Figure 19 - Summary of PII report of a SQL database	48
Figure 20 - Example of report from PII detection by category	48
Figure 21 - Encrypt Endpoint.....	48
Figure 22 - Http API to scrub lessons learned	56
Figure 23 - Dataflow mask assets	58
Figure 24 - Incident anonymization dataflow.....	60
Figure 25 - Mask Anomalies Dataflow.....	63

List of Tables

Table 1 - PII Categories	39
Table 2 - DSL Commands	41
Table 3 - PrivacyNet Services' Grouping and Mapping with Component's functionalities .	65

1 Introduction

1.1 Scope

This report presents the APIs and overall architecture of the PrivacyNet component of the CyberSANE Framework.

The deliverable focuses on the technical details of the multiple features designed, implemented and integrated. It starts with attribute-based encryption (ABE), then moves on to Privacy Models. Follows with dynamic processing to deliver a privacy framework that can deal with a diverse set of input formats and taxonomies. This deliverable lays out the groundwork and modelling language that allows for user defined configurations but comes “with batteries included” by providing sensible conventions and configurations that out of the box can deal with STIX 2.1 formats for Incident Data, Lessons Learned or Threat Intelligence. Detailing the APIs endpoints, their attributes and return format.

To clarify intent, the deliverable includes a few examples of remote calls, and the respective outcomes.

Afterwards, we present custom encoding algorithms and formats developed to support FPE (using AES-FF1) when anonymizing data that has strict length requirements, a regular occurrence inside legacy SQL databases schemas. These schemas have a variable encoding that require detailed and careful handling. This work also presents a way to deal with different encodings and provide user definable encodings through manual rune alphabet definitions.

On the network side, which is crucial for forensics and incident analysis network information, this report presents how the PrivacyNet supports NetFlow, ingestion or outputting it in a flat json format and capturing and processing of network data on a device interface, through raw packet captures. Supporter operations with packet captures, and how to anonymize and filter through DPI and relevant APIs follow.

Finally, we end with the report on the implementation of the privacy policy models and how they can be setup and driven by the required user defined rules as well as the integration with other 3rd party tools. During the integration, we dive deep into pipelining and how data processing occurs.

1.2 Contribution to other Work Packages

As mentioned above, the purpose of this document is to specify the list of services, APIs and functionalities of the Privacy Component, which serve as the main component for anonymizing information generated into the CyberSANE platform and information shared or gathered by external to the organization contacts and entities.

WP7 considers the business requirements which have been specified in the frame of “WP2 User requirements and Reference Scenarios” to define the tools and services required. From WP3 the study of current SOTA of threat information sharing or incident response report formats, inform what anonymization pipelines and rules are needed, as well as

common encodings, taxonomies and data models that are value added to be provided out of the box.

In WP2 (D2.4) a generic theoretical approach of web API of services and functionalities was layed out. That source is tweaked and presents in this document the new versions that were required to fill the implementation needs from integration work.

The tools and services from PrivacyNet support the CyberSANE Platform in the frame of “WP8 CyberSANE System Visualization, Integration, Deployment and Fine tuning”, but also the other CyberSANE Components.

1.3 Structure of the Document

The structure of the document includes the following main chapters:

- **Chapter 1** introduces the document, its scope, and the contribution to the other work packages;
- **Chapter 2** provides a high-level summary of the different tools in the support of the PrivacyNet functionalities, including the additional functionalities which have not yet been reported and have been integrated in the CyberSANE Platform;
- **Chapter 3** goes into lower-level APIs specifications, input parameters, processing and outputs, provided by the PrivacyNet component;
- **Chapter 4** concludes the deliverable and provides our plans for future activities.

2 PrivacyNet Functionalities

The following paragraphs introduce the tools integrated in the CyberSANE platform to support the PrivacyNet functionalities. PrivacyNet provides the privacy primitives to CyberSANE platform. In particular, it allows for the different CyberSANE components to perform the following functions:

- Anonymize structured data

The Generic Anonymization API allows for dynamically Anonymize files (text format) and / or text streams using traditional cryptography. In practice this endpoint permits the caller to define the input text stream, the encoding of said stream, the anonymization algorithm (MD5, SHA1, SHA256, SHA384, Blowfish, Text Generalizations, and others), the match rule (regex or other supported parser) and format the output should be encoded.

- PII Detection

The Personally Identifiable Information detector API, provides an endpoint that can search both SQL databases and known text formats for following categories Financial, Personal, National, Tech, Other.

Each category contains a set of attributes that are considered PII's for a given doc. For example, in the financial category, we have Bank Account Number, Credit Card Number, CVV and others.

This endpoint point can either report a JSON file with an array of location, PII Category, PII attribute and Rules that match, or an Excel file with the same information split among several sheets.

- Generic Encryption / Decryption

The Generic Encryption API allows for dynamically encrypting files and / or byte streams using traditional cryptography. In practice this endpoint permits the caller to define the input byte stream, the encoding of said stream, the symmetric encryption algorithm (AES, AES-FF1, AES-FFX or equivalent), the key size (128,192,256 bits) and cipher mode (CBC, ECB, etc) and format the output should be encoded.

- Privacy Policy Enforcement

The Privacy Policy API permits callers to setup a regular callback to notify a third-party system (through an HTTP Get or HTTP Post or DB Query) of the need to delete data that has expired it's need. This requires the callee to define both the callback URL, parameters; the frequency the check is performed; and the rule used to expire data.

- Attributed Based Encryption / Decryption

ABE API permits callers to dynamically parse a document and apply encryption or anonymization algorithms to only parts of the document. This allows the callee to redact partially whole documents, and have the output returned on the same format as it was submitted. Either JSON or well-known formats such as Excel or PDFs.

- Incident Data Redaction

The Incident Data Redaction API provides an endpoint that can process STIX 2.1 format for incident data and perform anonymization functions on specific attributes. For instance, IP Masking a generalization function that reduces IP significance by stripping the lower n bits and replacing them with a user provided mask.

- Lessons Learned Data Redaction

The Lesson Learn Redaction API permits callers to submit lessons learned in a pre-set format, and have it redacted using a list of rules that match any fields or values based on their definitions and removes or replaces said fields or values with null / predefined text / pseudo token. Then returns to the caller a document in the same format without the redacted PII's.

- Network Data Redaction

Redacting Network can operate with multiple input formats, either a PCAP format file that can be analysed with DPI tools for known protocols (TCP, UDP, http etc) or a flat json format with the relevant fields from NetFlow or SFLOW. PrivacyNet can process them both and output data in JSON format with a set of attributes or protocols redacted or filtered.

- Homomorphic Encryption

PrivacyNet provides the primitives to store data using homomorphic encryption with a particular scheme to allow homomorphic text search without decrypting the contents.

2.1 Internal Data Structures

Before we go into the details of the PrivacyNET architecture, it's essential to understand the principal database formats of the data structures used to power the system.

Such formats are mentioned several times in this document, when explaining the other components and are a key concept behind how everything is mapped.

In this section we present the Metago Object Format, the Event itself and the arc meta structure ReadBuffer.

2.1.1 Event

Inside PrivacyNET the events are parsed into Events. This structure contains a map from strings to generic interfaces which means all the keys are required to be strings and the

values of the map can be whatever type, interface, or primitive type. Internally pipelines work on arrays of events of type Event.

Below we present the go lang definition of the Event type.

```
// The structure where generic events are stored internally
type Event map[string]interface{}
```

Figure 1 - Event Struct

2.1.2 Metago Object Format

PrivacyNET is essentially a stateless flow processor, in the sense that multiple data streaming doesn't influence how other data streams are processed or handled. It's not a pure stateless system as we don't require actions or rules to be 100% stateless, they can aggregate self-contained statistical data for each run.

Being stateless forces data to be serialized. So to share data between rules or sent to another system the Metago object format is used. The objects consist of a structure with a header and an array of Events.

```
// MetagoObject - Structure ease serialization of events to JSON for client-server comms
type MetagoObject struct {
    Header map[string]string `json:"header"`
    Events []Event           `json:"events"`
}
```

The events are multiple entities of the structure defined on the previous subsection 2.1.1.

The header contains information specific to events on the payload, and by default it includes the following fields:

```
"header": {
    // Unique agent identifier uuidv4
    "agentId": "metago-4e60b304-bb8c-480a-977d-c780f315d430",
    // CPU Arch
    "architecture": "amd64",
    // Event structure
    "format": "metago",
    // Hostname / Device Name
    "host": "MacBook-Pro.local",
    // Index where data should be stored
    "index": "test_suricata",
    // Operating System name
    "os": "darwin",
    // Source where the events come from
    "source": "test/data/json_row.json",
    // Sourcetype name
    "sourcetype": "suricata",
    // Timestamp field
    "timefield": "timestamp",
    // Time parser string
    "timeformat": "YYYY-MM-DDTHH:mm:ss.sssss%Z",
```

```
// Timezone
"timezone": "Europe/Lisbon"
}
```

2.1.3 Source

The interface for all inputs, any new input component needs to at least provide the methods below to work seamlessly with PrivacyNet rules and sinks.

Each method is preceded with a summarized description of its purpose.

```
type Source interface {
# A human readable URL that represents the source of data
    URL() string
# The main method that collects data and send it downstream through a ReadBuffer Channel
    Monitor(in chan *ReadBuffer) chan bool
# A callback to be invoked by the sinks when the last event has been processed, to give the source notice to
close any dangling resources, such as database handles or file handles
    NotifyLastEvent(e *Event)
# A callback to be invoked by the core system, to let give the source the chance to exit cleanly, by releasing
acquired locks and resources, to stop the influx of data and keep the status consistent
    Quit()

... other methods provided by the internal system, that are shared between different sources
}
```

2.1.4 ReadBuffer

This is the main container used for data processing. Each pipeline will receive streams of data through a channel of ReadBuffer objects. All objects are initialized prior to handling by the program, to avoid null dereferences.

Inside ReadBuffer we have an array of bytes that can store raw data in any format. A source referencing the place where data comes from, an array of events that store transformed data, and an array of errors that keep track of any issues encountered during the runtime processing of the current ReadBuffer.

```
type ReadBuffer struct {
    Bytes []byte
    Source Source
    Events []Event
    Errors []error
}
```

2.2 PrivacyNET system architecture

In this section we present the high-level description of the PrivacyNET component, how the main blocks talk to each other and how the information flows internally. Details on how to communicate with other components are presented in the next subsections.

Below we have the general architecture for PrivacyNet.

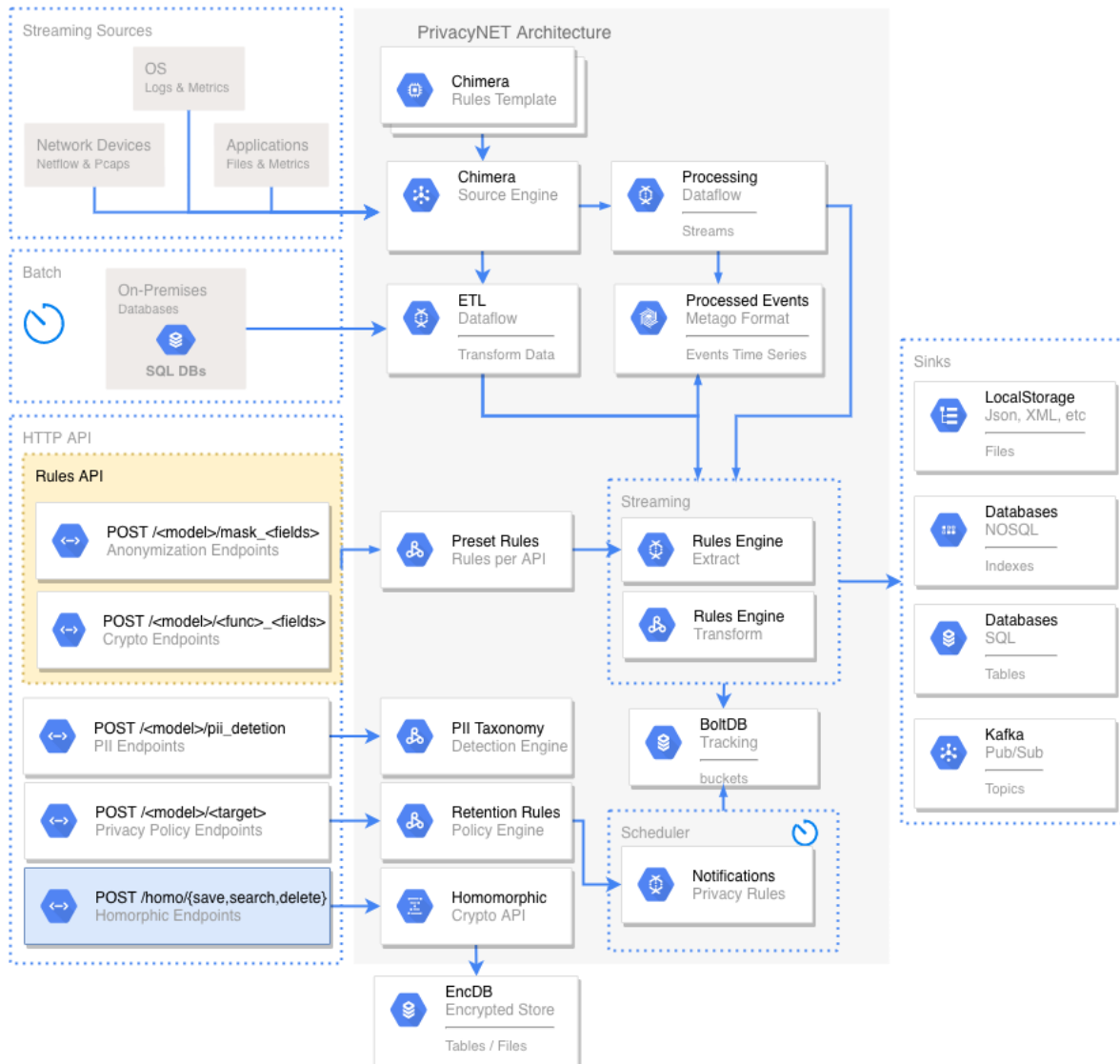


Figure 2 - PrivacyNET High Level Architecture

2.2.1 Configuration Files

There are two configuration files config.toml and rules.toml. Config is used for setting up sources and outputs, and rules to setup the transformations required to parse sources into the internal Metago format.

2.2.1.1 Config.TOML

Config used the TOML format where sections are demarked through stanzas ([name]) with property then being mapped into a map like structure where the attribute keys are strings (in this case) and the values can be any of the following common objects:

Strings, floats, Integers, Booleans, Arrays / Lists, Maps.

Below is an example of a configuration file, that reads the file on the relative path `"/test/data/dnslogs.txt"`, and applies the relevant rules, `dns`; `str`; `trim` and `debug`.

```
[file.in]
Path="/test/data/dnslogs.txt"
BatchMode=true
NoTracking=true
Rules=["rex.dns","replace.str","string.trim","outputs.debug"]
[outputs.debug]
Urls=["https://example.org/path/"]
```

Untangling this configuration, from the outputs side, it will parse the URL and assume based on the `https://` start that it will contact the machine at `example.org` and make an async HTTP Post request to the location `/path`. The body of the HTTP Post will be a JSON document, with the METAGO object format.

The full spec for Config.TOML is available on Annex I.

2.2.1.2 Rules.TOML

This configuration file follows the same TOML format as Config, but it is where processing rules are declared. Each rule should be defined following the stanza convention, which requires the stanza to be named as [`<rule type>`].`<name>`]. Rule type defines the actions that will be taken, and the name is how the rule is stored internally, so that you can refer to them in the inputs' Rules attribute.

Considering the example from before, let see how rules are declared and go through at they do.

```
[rex.dns]
Patterns=["(?P<timestamp>\d+\d+\d{4} \d+:\d+:\d+ ..) \w{4}
\w+\s+\w+\s+\w+\s+(?P<direction>\w+)\s+(?P<client_ip>\d+.\d+.\d+.\d+)\s+(?P<xid>\w+)\s+(?P<queryresponse>[
\w])\s+(?P<opcode>[ \w])\s+(?P<flags_hex>\w+)\s+(?P<flags>[
ADTR]{4})\s+(?P<response_code>\w+)\s+(?P<query_type>\w+)\s+(?P<query>[^\n]+)"]
# Optional
# Field on which to apply the regular expressio, if nothing is given it will be checked against
# the internal Bytes on the ReadBuffer
Field="_raw"
# Optional
# Since: 1.1.48
# Allows for the grep -v logic where the regex is inverted
Invert=false
# Optional
# Since: 1.1.48
```

```
# Specify the run mode, filter the Events or Extract Fields, by default extracts fields
Filter=true
```

This defines a regular expression rule, with has a set of attributes. Patterns being the only mandatory one, and which should be an array of strings that map to a regular expression.

“rex.dns” – A rule to extract the DNS request attributes through a regular expression

“replace.str” – replaces (<number>) from a DNS request with the regular “.”

“string.trim” - removes the leading and trailing “.”

```
[replace.str]
Field="query"
Regex='(\d+)'
Value="."
[string.trim]
Fields=["query"]
Op="trim"
Args=["."]
```

2.2.1.3 Configuration Folders

PrivacyNET can be configured by dropping rules.toml and config.toml files into two folders relative to the installation path. First the folder “./default” will be scanned for these files, afterwards the folder “./local”, if there are configuration on both directories for the same files, then a merge between the stanzas present in both files occurs. In this merge the configuration from the local folder takes precedence and override any stanza with the same name in the default folder. Finally, if there are user defined specific files that are different from the ones already present in the default and local folders, these files are read and their content is merged with the current configuration, these user defined files take precedence and override previously configured stanzas.

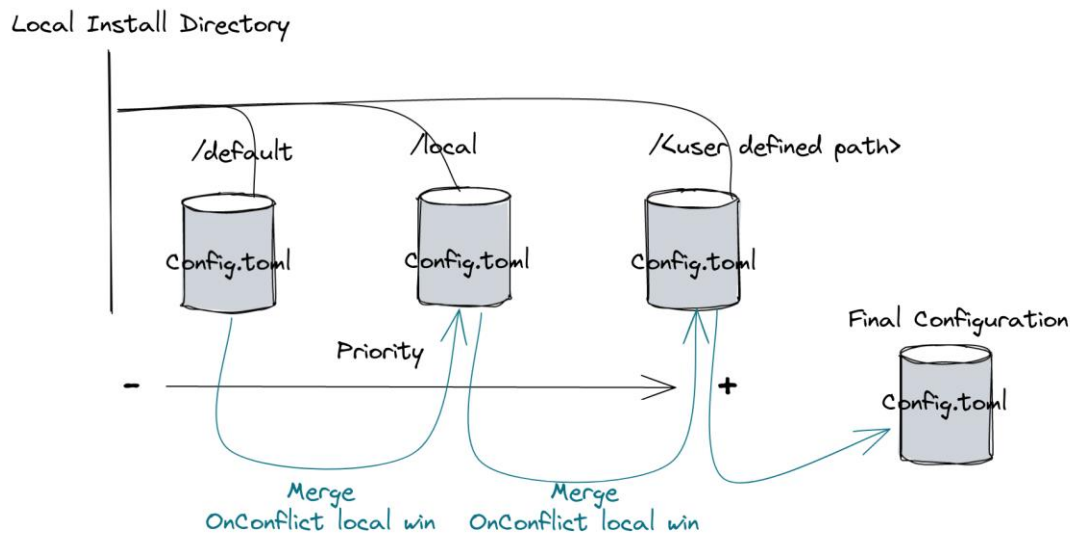


Figure 3 - Configuration files

2.2.2 Pipelines

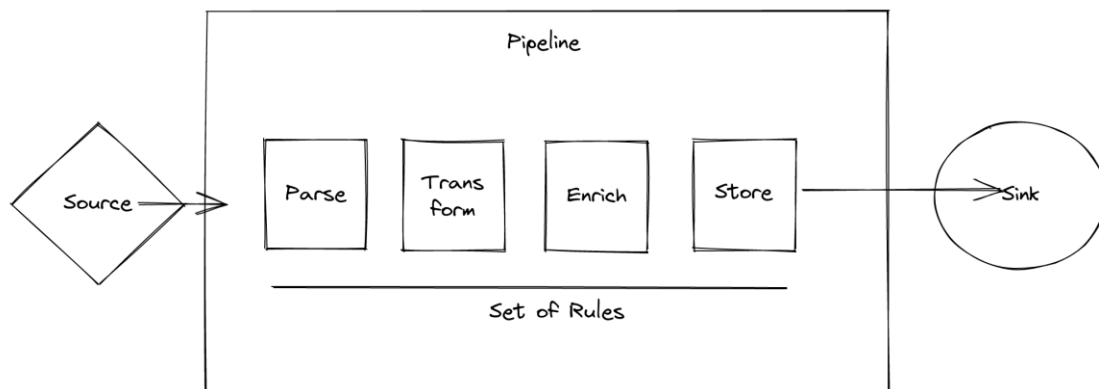


Figure 4 - Pipeline Generic Example

Pipelines are the central concept of this component; they represent the set of rules a given flow should have. They map a source, generically any way to receive data, with a list of rules that run-in sequence on a batch of events. Those rules manipulate each event and can perform any function from the set presented on the later subsections. They can be viewed as ways to mutate and transform data, to match the desired effects by the user. Optionally the pipelines have outputs (also called sinks), resulting in the data at the last transformation being stored in a temporary or permanent storage. Example of such storage would be SQL databases or plain text files.

2.2.3 Sources

Sources are any connectors that retrieve data from external systems or read data from sockets or local files. In the next subsections we present the most common sources and detail their functions.

2.2.3.1 HTTP API

The main entry point of data into PrivacyNET is HTTP API, it allows for simple RESTful calls. These endpoints can be broken down into 5 main categories: anonymization, cryptography, PII detection, privacy policy and homomorphic encryption. The details for the features will be presented on the next chapter. Here we will present a generic view of the HTTP input features and their relation to the rest of PrivacyNET.

The original intent of an HTTP input would be to allow for the dynamic definition of web services that could help users map predefined paths to a pipeline and get back the results from applying that pipeline rules to the payload sent on the original request.

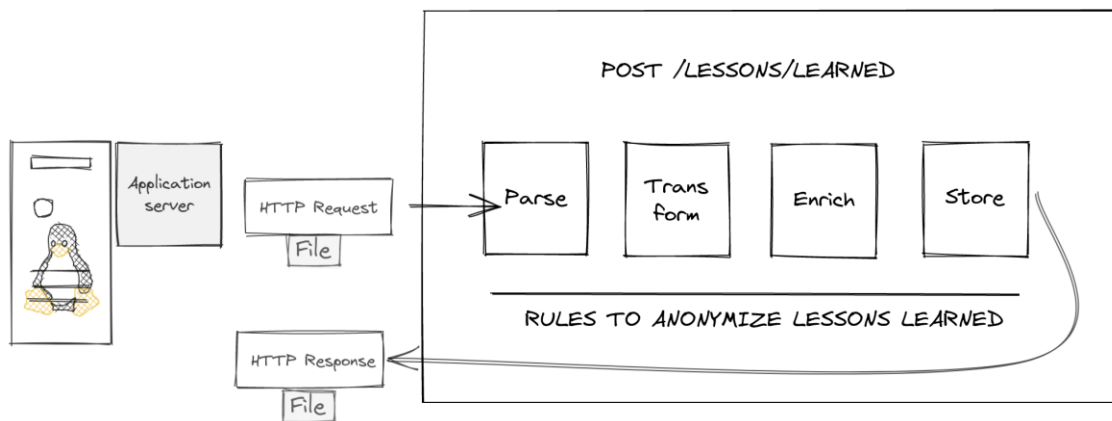


Figure 5 - Traditional Http Request API

However, to improve the performance when dealing with large files, and to simplify the architecture when deploying on Docker containers or inside a Kubernetes cluster. An alternative mode of operation is made available for PrivacyNET. One where the callers can invoke the HTTP services but instead of sending the raw bytes of the file to be processed (encoded in whatever format we could agree upon, that would be simultaneously supported by the caller and the callee), they provide only the path of the file to be read on a common volume. This has the merit of avoid the multiple serialization and deserializations costs through multiple http requests but imposes the restriction of a shared storage between both components.

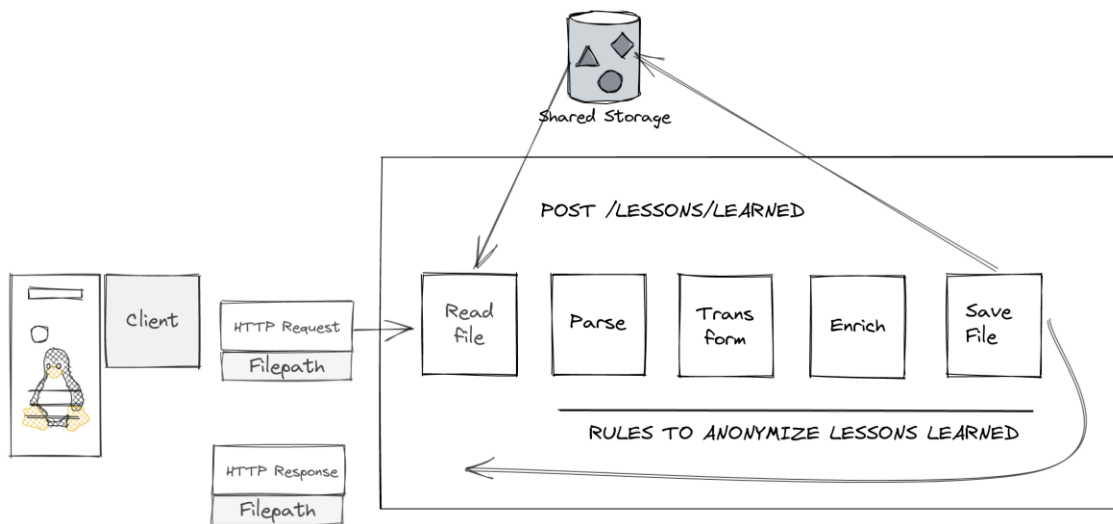


Figure 6 - Integration through local storage

2.2.3.2 Streaming Sources

Monitoring files is the most common streaming source for PrivacyNET, this is relevant when ingesting log files, for instance to scrub PII data from them, prior to save on permanent storage.

File sources are the bread and butter of any log collection agent, they can be in multiple formats, plain text, json, xml, syslog and any other text format there is. This component is agnostic to format, but if you want to properly parse time and event attributes, it's up to the user to setup the processing rules. By default, datetime is extracted for each parsed event using complex and large regular expressions, but this can be overwritten by user configuration. Time zones and encodings are also configurable, we favour convention over configuration, so it's recommended to keep the default values omitted and only specify what needs to be changed.

Below we present an example configuration spec for reading the file "in.csv".

```
## FILE ##
# Stanza example for monitoring files
[file.csv]
# The filepath to monitor, can be absolute or relative to work dir
Path="in.csv"
# Array with the names of the rules to be applied to this input
Rules=["rex.csv", "ff1.dstip", "hash.srcip"]
# Array of outputs where processed events will be sent after they go through the pipeline
# IMPORTANT: At the moment this array can only have 1 output, having 2 or more will cause random lockups
Outputs=["kv"]
# File encoding, by default utf-8 if nothing else selected
# Full list available at docs/encode_formats.md
Encoding="utf-8"
# Don't keep track in the internal db of the current offset that's already processed
NoTracking=false
```

```
# Don't monitor file forever, just read once until EOF
BatchMode=false
# String Specify the characters that will be considered for an EventBreak, by default "\n"
EventBreak="\n"
# Whether of not this input is active
Disabled=false
# Random chars that can be used to bypass previously stored states/offsets for a given file url
# Useful when the file has been rewritten and it's required for it to be processed from 0 again
Salt=""
# Boolean to specify if the input should be threats as a single file or a folder and all files inside should be monitored
Folder=false
# Max number of bytes to read at a time from the file
BatchSize=65536
# Since: 1.1.49
# Disable event breaking, this will present all the read bytes in the current batch.
# In combination with BatchSize it can be used to read the file in one go, this is useful to process json files
NoEventBreak=false
```

2.2.3.3 Batches

On the off chance that files need to be processed once, it's also provided a mechanism to read file in a single run. In the previous subsection, the property BatchMode was disabled as it was set to false. If this property is set to true, then the file will only be read once, and further chances to it, won't be read by PrivacyNET unless the component is restarted. Running inputs in BatchMode is mostly useful for testing purposes or for running the tool on the command line.

In CyberSANE BatchMode should always be disabled when running as a service in production environments.

In special inputs such as the traditional SQL databases, batches can take a hybrid form. They can run at a given schedule, using CRONTAB like definition with an extra position at the beginning to define the seconds recurrence rule.

```
_____ second (0 - 59)
| _____ minute (0 - 59)
| | _____ hour (0 - 23)
| | | _____ day of the month (1 - 31)
| | | | _____ month (1 - 12)
| | | | | _____ day of the week (0 - 6) (Sunday to Saturday)
| | | | |
| | | | |
| | | | |
* * * * * <command to execute>
```

2.2.4 Sinks

The outputs that come bundled with PrivacyNET are the called sinks, for their ability to receive data that has been processed and converted into the internal object format. Sinks are the destination where data can be stored and saved. They range from the traditional SQL databases to new NOSQL trends, such as Elastic Search. But can be as simple as a text file stored on a local disk. In the next subsection the relevant sinks are presented and detailed.

2.2.4.1 SQL Databases

The SQL DB sink executes an UPDATE or INSERT INTO sql query on the selected database. The query is loaded as a prepared statement to protect against SQL injection on the arguments, which are specified by the users or from the data being processed.

Below we present the full specification of attributes for the configuration of this stanza.

```
##### DBOUT Save to DB #####
# Execute an Insert or Update statement constructed from a SQL template with optional Params that will be
# retrieve from each Event
[dbout.cadev_out_sim]
# Database definition where to connect to
DB="sitam"
# Truncate the output if db column can't store it all, instead of letting the DB deal with it
Truncate=false
# Prints the SQL instead of executing updates implies KeepStatus = false
DryRun=false
# Prints the prepared statement as SQL query instead of placeholders with args
DrySQL=false
# Params to be used as query parameters
Params=["Cliente","ClienteNome","ClienteNIF","ClienteDataNascimento","ClienteMorada","ClienteTelefone","C
lienteEmail","__STATUS__","__DATE__","SimulacaoId"]
# SQL to be executed
Query="UPDATE [dbo].[Simulacoes] SET [Cliente] = ? , [ClienteNome] = ?, [ClienteNIF] = ?,
[ClienteDataNascimento] = ?, [ClienteMorada] = ?, [ClienteTelefone] = ?, [ClienteEmail] = ?,
[estadoAnonimizacao] = ?, [dataAnonimizacao] = ? WHERE [SimulacaoId] = ?"
# Whether or not to keep tracking of the processed rows
KeepStatus=true
```

2.2.4.2 Elastic Search

The Elastic Search data sink inserts new documents into an existing Elastic Search database. Inside the metago format (2.1) the events are stored inside an array, for each event a new document will be created on the target Elastic Search database.

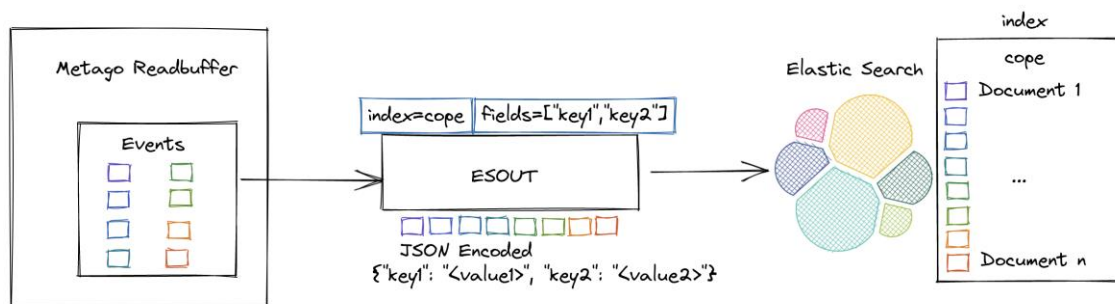


Figure 7 - ESOUT output to elastic search

The esout stanza, iterates over the events inside a ReadBuffer, converts them into JSON selecting only the fields defined on the configuration file, and asynchronously and using multiple threads sends them into an ElasticSearch database. Inside Elastic Search, indexes have the same roles has databases inside a tradicional SQL database, and documents are the equivalent to the column/rows inside a SQL table.

The current spec for using this stanza is:

```
#### ESOut ####
# Saves the current pipeline result to an elastic search database
[esout.savetoelastic]
## Required ##
# List of fields to save to elastic
# this will save {"name": <namevalue>, "age": <agevalue>}
Fields=["name", "age"]
# ES index
Index="main"
Host="EShost"
Port=9200
## Optional ## authentication
# User and Pass
User=""
Pass=""
```

2.2.4.3 URLs

Sinks based on URLs can be both local and remote, they provide a concise way to save data in a pre-defined format. The set of common formats out of the box are:

Format	Description
raw	Writes Readbuffer.Bytes directly without any conversion or encoding. Delegates the responsibility for encoding into the pipeline rules that come before.
json	Formats the ReadBuffer.Events in JSON as an array of events

	# Example: [{"key": "value1", "other_prop": 1}, {"key": "value_event2", "other_prop": 2}]
kv	Formats each event in ReadBuffer.Events with a Key Value format split by newline "\n" character. # Example: key="value", another_key="value2"
csv	Formats the ReadBuffer.Events in CSV format. By default uses "\n" newline character to split the event, "," to split between multiple columns and '"' as the encoding character (begin " + Value + " end) to specify a string value (normally only required for values that contain the special characters used for the CSV format, as mentioned on the previous sentence). # Example: key,another_key "value","value2"
custom	See detailed explanation below

By convention the start of the URL defines the template to encode the ReadBuffer information. When convention is not enough users can fallback specifying a Custom format and use the Template attribute which will be sourced to encode each event inside ReadBuffer.Events. The \$var_name is used as a key on the Event being processed and replaced with the string representation of the value found inside the Event. If no such key is present, it will fall back to an empty string.

To save the data locally there are 3 possible options, file:// which receives a local path where the kernel syscall write will be used to write or append the file. The pcap:// format will save the raw network packets that should be inside ReadBuffer.Events on the "_packet" attribute. This assumes a network_interface used as source and the value of the "_packet" key inside the Event is filled with the proper captured packet data with packet bytes and metadata. Lastly it is possible to use stdout to print to the terminal's output where the tool is running.

Below we have the spec for the output stanza:

```
# Output stanzas to be used save data to a url
[outputs.kv]
# Format to write the output in
# "raw" - Writes the bytes was they arrived during the pipeline
# Useful for binary data transfers or pcaps
#
# "json" - Formats the ReadBuffer ([Event]) in json as an array of events
# [ {"key": "value1", "other_prop": 1}, {"key": "value_event2", "other_prop": 2}]
```

```
#
# "kv" - Formats the ReadBuffer ([]Event) in key value format
# key="value" other_prop=1
# key="value_event_2" other_prop=2
#
# "csv" - Formats the ReadBuffer ([]Event) in csv format
# key,other_prop
# value,1
# value_event_2,2
#
# "custom" - Formats the Events using a string template defined on the Template attribute
Format = "custom"
# Template to be used on custom output formats
Template="$name is the best journalist in the whole $place"
# Magic URLs that point to the right output
# Supported Formats
# "tcp://host:port" - Open a TCP connection to <host> at <port> sends the event bytes there
# "udp://host:port" - Open a UDP connection to <host> at <port> sends the event bytes there
# "metago://host:port/path" - Uses HTTP with METAGO event encoding to <host>:<port>/<path>
# "metagos://host:port/path" - Uses HTTPS with METAGO event encoding to <host>:<port>/<path>
# "file://local_path" - Write to a local file
# "pcap://local_path" - Writes a pcap file, only to be used with network packet captures
# "stdout://" - Write to STDOUT
Urls = [ "stdout://" ]
## OPTIONAL FIELDS ##
# Only useful for file outputs, values can be "append" or "write"
Mode="append"
```

2.2.4.4 Files

Saving data into files, follows the same rules as specified on the 2.2.4.3.

2.2.5 Core

The core is the engine for PrivacyNET, the place where the mapping between inputs and rules happens, and where the event pipeline churns along the events passing through. In this section we present the main components in the PrivacyNET and explain how data flows through them.

2.2.5.1 Source Engine

When the PrivacyNET starts it will read the configuration files, parse them into 4 types of entities. The inputs, the outputs, the rules, and system configurations. The inputs are the sources defined in 2.2.3.

The source engine is responsible to connect both the setup inputs with the outputs by stretching out the rules that were defined in the configuration files. As specified on the configuration file config.toml all the inputs receive a list of rule names. The source engine is responsible to map those names into the configured rules on the rules.toml stanzas.

In the next subsections we will describe the different configurations starting with the system configs, will go into detail about the metago object format as well as the rules, the templates, the presets and other specific functionalities that PrivacyNET provides.

2.2.5.1.1 System Configurations

Inside of the system configurations we have 3 stanzas types that define different entities will start off with explaining the database stands up which allows the users to define connections to standard SQL databases disconnections receive the usual host port user passwords and a few other custom parameters that these databases might support.

Next, we present the source types. Source types are an abstraction on the format for the data being read on the source types we define stuff such as encoding's time format and a way to parse times from whatever has been reads this allows the user to define their local time string parsing format to match whatever that computer his endpoint or the server where is collecting data from is set up with. parsing time is one of the critical steps for each event processing will go more into detail in the later subsections.

Lastly, we outline the system configurations.

- **Databases**

```
# DBConfig - struct to hold database configurations
[db.sitam]
## Required Fields ##
User="notroot"
Pass="youwishyouknew"
Host="hostofdb"
# Database driver to use, supported:
# mysql
# psql or postgres
# mssql or sqlserver
Driver="mysql"
Port=3306
Database="sitam"
# Since 1.1.51
# Defines the max number of connections to hold in pool 0 for unlimited
MaxConn=0
```

Database stanzas start with DB then we have a dot to split between the type and the stanza name and finally we have the stanza name.

In this example we are setting up database with the name sitam. The required fields for any database configuration include user pass host driver ports and database optionally users can also define a Max connection with Speech specifies the Max number of connections to old on the pool for this database if nothing is defined or zero which is the default configuration is specified the pool can have unlimited number of connections which are obviously limited by the resources of the underline system where the agent is running.

These database stanzas can then be referred to from either inputs or rules for instance the DBin stanza, that specifies and input from a db where a SQL query is going to be run on predefined Cron recurrent expression. Also inside rules.toml configuration file we can refer

to this database configuration from rules types that operate on databases for instance the DB lookup rule.

- **Sources Types**

```
# SourceTypes
# Since: 1.1.0
# SourceType is a string that defines the type of the source, and configures the time parser to be used
# as well as the timefield to be considered as the time of the event
[sourceypes.sysmon]
# Required Fields
# Format of the event should be encoded when sharing with other systems
Format="METAGO"
# Field to be considered as the time of the event
TimeField="timestamp"
# Time parser to be used to parse the time field
TimeFormat="YYYY-MM-DDTHH:mm:ss"
```

The source type stanzas are where we define the meta attributes for event parsing. In here we associate a name as shown above in the example with the source type for Sysmon, with the format for event sharing, the field to be considered as the time for the event and the parts of string to be used when parsing the event time field.

source types are useful abstractions to reduce the boilerplate information since we are often going to encounter formats that we have already parsed in the past it's useful to be able to refer to previous configurations and not repeat on each input stanza the same format time field and time formats.

- **System Configuration**

```
# Stanza for internal system configuration
[system.web]
# Disables the web server API
Disabled=true
# Since: 1.1.57
Port=8080
```

Inside the system stands there we can configure for the time being only two things whether the web server for the agent is going to be launched when the service starts and the port where that web service should be listening on. This web server is important for the service to be able to be orchestrated remotely but when it's run as a command line tool, we generally want the web service to be off for testing purposes since this configuration imposes a lock on whether the service should exit for processing shopping all the data that it had gathered on the inputs. If the property "Disabled" is set to true in the *system.web* stanza, and if we only have inputs that don't require active monitoring like a file input with BatchMode=true, the tool will exit after all events have been processed.

2.2.5.2 Rules Engine

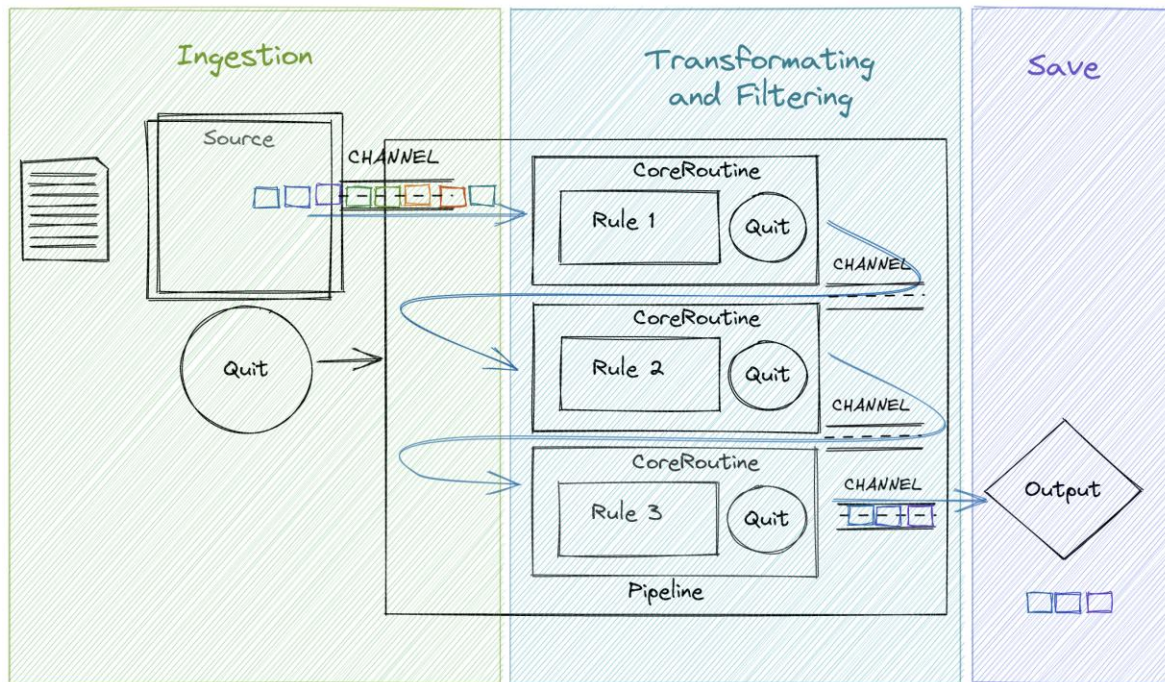


Figure 8 - Rules Processing

Rules are the bread and butter for the core system, they live inside pipelines and form a daisy chain where the output of the first rule is passed on to the input of the next one until the pipeline is finished. At the end the result is sent to respective output that will be connected to the pipeline.

To better understand what rules are and what types of rules exist, we need to start with the ingestion part. Ingestion occurs at the sources, it takes a set of documents, HTTP requests, database rows, or anything else that matches a source or an input, converts the data inside into a ReadBuffer object. ReadBuffer is then sent streaming into the pipeline. The pipeline makes all events inside of that ReadBuffer go through the rules in the daisy chain. Each rule can manipulate events on the individual level between them, the events can be filtered, augmented, and created. For instance, regex rule can select only events that match a given regex on a predefined field.

Each rule runs in its dedicated coreroutine, which runs on it's one dedicated goroutine. In effect all rules inside a pipeline can run concurrently and in parallel.

Rules share with pipelines the architecture for notification and asynchronous processing. Both are based on the CSP [2], rely on channels and communicating through messages without sharing memory. Rules contain one channel for input, one for output and another for exiting. The input and output ones have been previously explained. The Quit channel's purpose is to notify the rule that the main engine is requesting its orderly termination as soon as possible. The rule is then responsible to stop consuming data from the input, processing the ReadBuffer that it might be processing, and exiting freeing any resources it might have acquired.

For those familiar with POSIX [3], the first notification on a rule to quit has semantics similar to those of sending a SIGINT signal to a running process, and the second notification has the semantics of a SIGKILL.

Under the scene rules are processed following the algorithm:

1. Receive a ReadBuffer from the input channel
2. Create a new ReadBuffer to save the output
3. For all events in the ReadBuffer do
 - a. Extract the value for the date property from the event
 - b. Apply the month generalization on the value
 - c. Create a new event with all the properties copied from the original event
 - d. Replace in c) the value b) of property "date"
 - e. Save the new event in a new ReadBuffer 2)
4. Send ReadBuffer 2) into the output channel

2.2.5.3 Rule Types

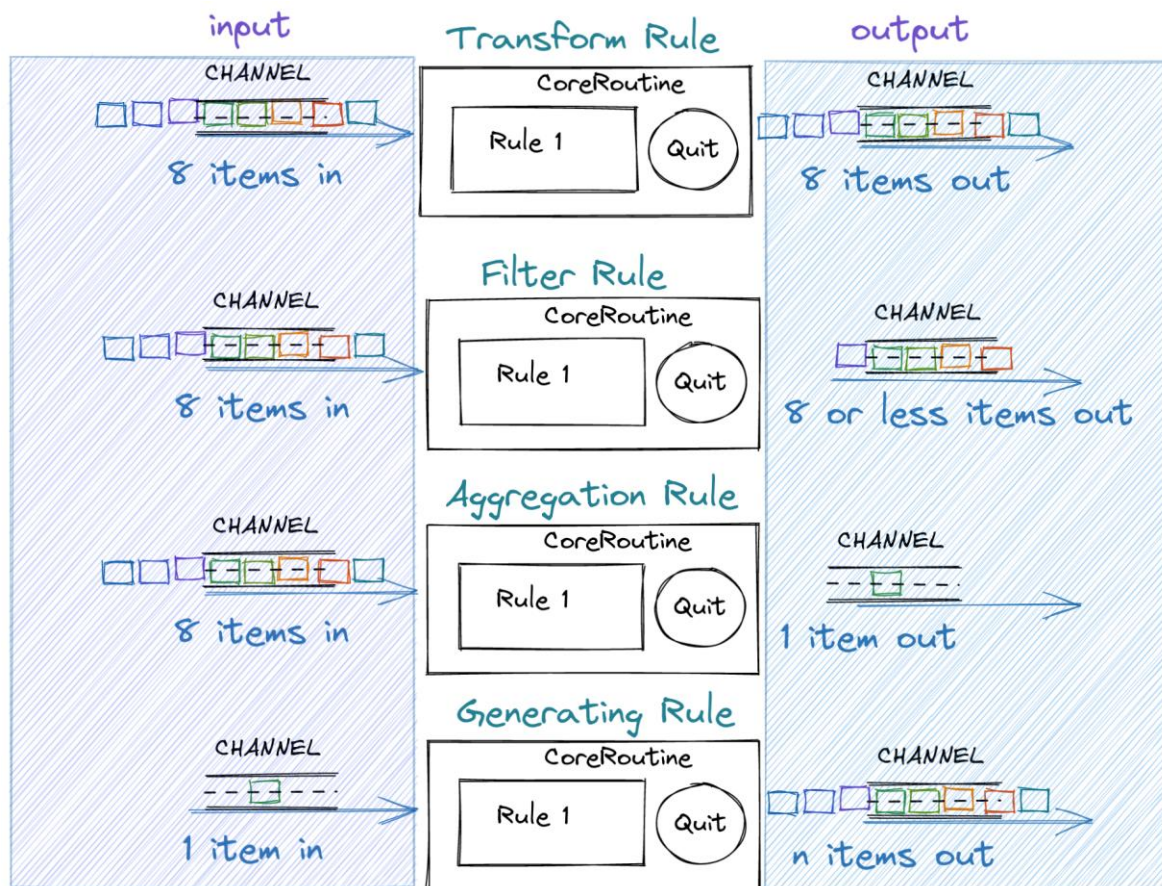


Figure 9 - Rule Types

2.2.5.3.1 Transform Rules

The most common type of rule is a Transformation rule, it will process an event and mutate some of its properties. To exemplify the type of rules we are writing about please check the Bucket rule spec below, later we will go through how it works.

```
##### BUCKET #####
# Takes a list of fields and performs date rounding to
# one of the following Durations
# "second", "minute", "hour", "day", "week", "month", "year"
[bucket.bin_riscos]
# List of fields to be generalized
Fields=["BirthDate","LicenceDate"]
# "second", "minute", "hour", "day", "week", "month", "year"
Duration="month"
# Format of dates / time to be processed
# follows clever format, 06-year, 01-month, 02-day etc
Format="2006-01-02 15:04:05"
```

The bucket rule applies a generalization of a date into a less specific date. Let's say we start with the following event:

```
{"date":"2020-10-04 15:12:04", "name":"Bill Taylor"}
```

And the following rule:

```
[bucket.date]
Fields=["date"]
Duration="month"
Format="2006-01-02 15:04:05"
```

The outcome would be the following event:

```
{"date":"2020-10-01 00:00:00", "name":"Bill Taylor"}
```

2.2.5.3.2 Filtering Rules

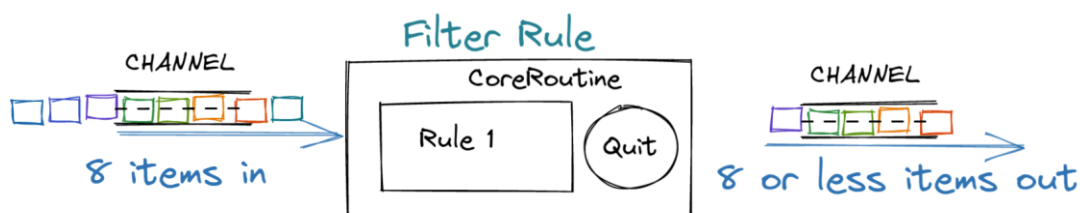


Figure 10 - Filter Rule

Unlike transform rules, where the input cardinality and the output cardinality should be the same, filter rules can reduce the cardinality of the number of events. In lay terms this means reducing the number of events. If n events are ingested, at most n are returned.

As an example, we present how the REX (grep) rule works, consider the following spec:

```
#### REX Regular Expression ####
[rex.myregexes]
# Required
# List of patterns to process by order on each event going through the pipeline
Patterns=["(?P<name>)[^,]+", "can't stop (?P<program>[^ ]+) from crashing and (?P<sideeffect>[^ ]+)"]
# Optional
# Field on which to apply the regular expression, if nothing is given it will be checked against
# the internal Bytes on the ReadBuffer
Field="_raw"
# Optional
# Since: 1.1.48
```

```
# Allows for the grep -v logic where the regex is inverted
Invert=false
# Optional
# Since: 1.1.48
# Specify the run mode, filter the Events or Extract Fields, by default extracts fields
Filter=false
```

The rex rule applies a list of regular expressions to a given field, only the first to match will extract any new fields. Fields can be extracted using named captures or traditional regex grouping (?P<name>)[^,]+) -> would extract to the current event {"name": "<captured_text>"} and ([^,]+) would extract to the current event {"0": "<captured_text>"}

Optionally rex can also filter events, selecting only the ones that match any of the user configured regular expressions.

Let assume we have the following rule:

```
[rex.myregexes]
Patterns=["(?P<name>)[^,]+",]
Filter=true
Field="message"
```

Applied to the following list of events:

```
[
  {"message": "John, its late"},
  {"message": "I am late"},
  {"message": "I am late again"},
  {"message": "What is going on?"}
]
```

Would output:

```
[
  {"message": "John, its late", "name": "John"}
]
```

Only the first event matches the regular expression, and as such only that event is copied to the output and named extraction is applied.

2.2.5.3.3 Aggregation Rules

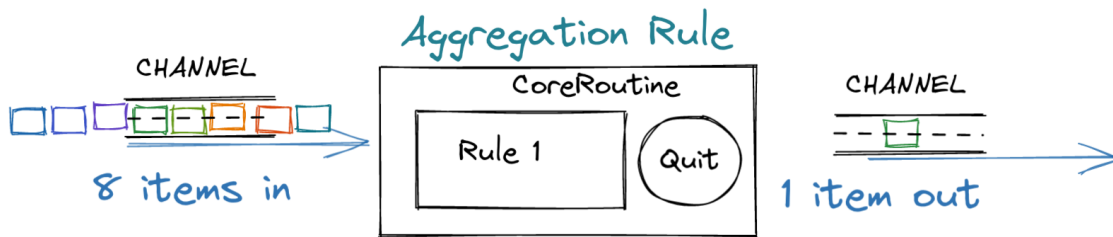


Figure 11 - Aggregation Rule

Aggregation rules are used to reduce the number of events, they can be thought of as a way, to reduce the cardinality of the input set. Considering we would like to calculate the average value of a given metric, it would take the input set of any size and produce a single value representing the average.

Output or recoding rules are also considered under this category, as an example we offer to the reader the particular case of the Excel Output rule, which is presented below:

```
#### Excel Out ####
# Converts the current list of events into an Excel xlsx file format
# and saves it into the current rb.Bytes
[excelout.savefile]
# Optional
# Sheet name on the saved Excel file by default Sheet1
SheetName="Sheet1"
```

This rule is special in many ways, it doesn't actually change the cardinality of the set, both input and output sets will have the same number of elements. But the operation it performs, does indeed belong under the aggregation category. It will process a set of input events, store them all in a single sheet inside an Excel file, then it saves that Excel file into the ReadBuffer.Bytes attribute. This effectively reduces to a single entity the set of inputs. Note this rule doesn't persist the output in disk storage, it merely serializes into memory. For persistence, the pipeline should require a new File Output. This type of separation of concerns allows for greater composability of rules and pipelines. And reduces the limitation imposed by the minds of the original programmers.

2.2.5.3.4 Generating Rules

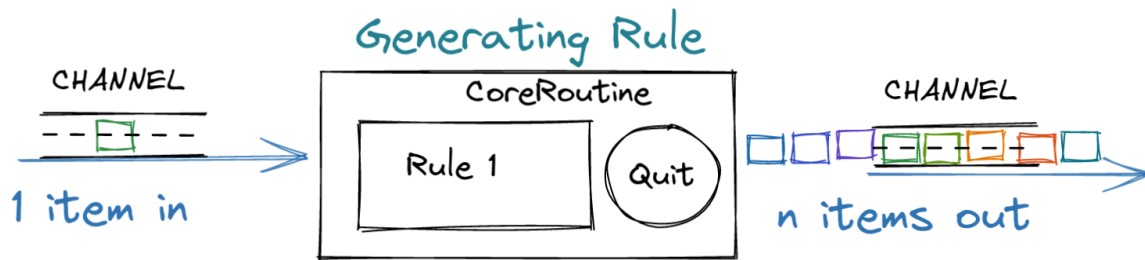


Figure 12 - Generating Rule

Lastly, we describe the generating rules, they perform the inverse effect of aggregation rules. They take a single element and generate a set of events. A common example of a generating rule is a CSV parser. It takes raw bytes or a string and parses them into multiple events, for each line inside the CSV a new event will be created. Below we present the CSV parser spec:

```
##### CSV #####
# Parse event bytes from CSV to Event
[ csv.in ]
# Should we use the first line as headers?
Headers=false
# Alternative to define the column name
Fields=["column1","column2"]
```

Now consider the following input in ReadBuffer.Bytes:

```
id,year,month,day,hour,minute,second
1,2020,1,25,3,44,57
2,2020,1,25,3,44,52
3,2020,1,25,3,44,48
```

And the following rule spec:

```
[ csv.in ]
Headers=true
```

This would generate the following ReadBuffer.Events on the output:

```
[
  { "id": 1, "year": 2020, "month": 1, "day": 25, "hour": 3, "minute": 44, "second": 57 },
  { "id": 1, "year": 2020, "month": 1, "day": 25, "hour": 3, "minute": 44, "second": 52 },
  { "id": 1, "year": 2020, "month": 1, "day": 25, "hour": 3, "minute": 44, "second": 8 },
]
```


2.2.5.4 PII Detection Engine

The PII detection engine supports multiple techniques to identify PII inside semi-structured documents. It goes from the simplistic regular expression approach to an advanced ML detection [4]. Since either regular expressions or NERC [4] can lead to high rate of false positives a DSL [5] for PII detection has been added to the PII detection engine. This DSL allows for additional restrictions that enable white and blacklisting rules based on meta information or context. Meta information can be column type specification from SQL databases, filenames, or source types. By exploiting that meta information, we can reduce the false positive rate, and tailor the process to document types. If the format of a document is known before hand, detection rules can be tweaked by users to improve detection.

A normal workflow includes the following steps:

1. Read input data (either from files, URLs, or databases)
2. Define rules to extract dynamic fields
 - a. Either through regular expressions or custom-made parsers (key=value, csv, xml, json, etc)
3. Define the rules to transform fields (either add, remove or mutate in place)
 - a. Can anonymise, encrypt, decrypt, refactor data
4. Write output report to an excel file

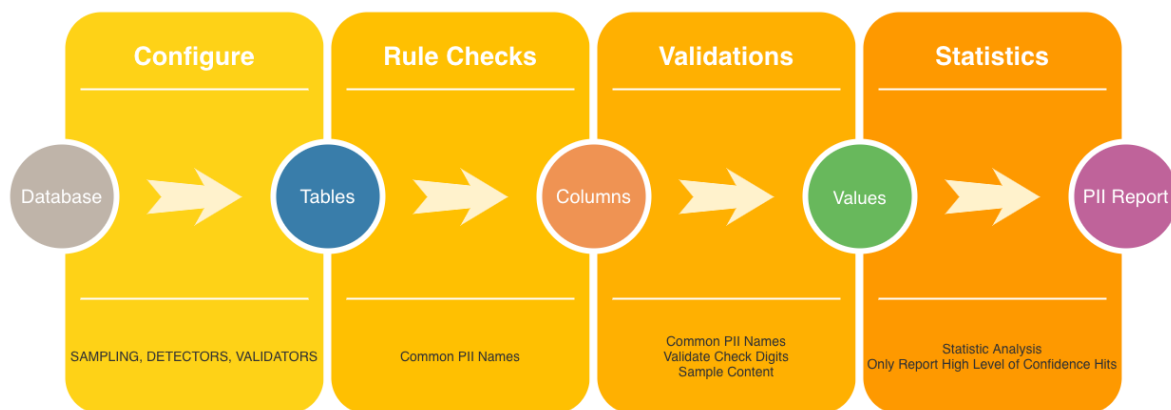


Figure 13 - PII Process

2.2.5.4.1 Regular Expressions

As example we provide the regular expressions used to detect mails, zip codes and phone numbers. Internally we categorize the regular expressions into distinct groups such as financial (banking related), personal (of the individual, age etc), national (related to national id cards), technical (ips, urls, macs), and other (anything that doesn't fit the other categories). Inside each category a set of regular expressions detect a variety of PII. In total about 90

regular expressions are provided by default to be consumed inside PrivacyNET. More expressions can be user defined and added manually to a running system.

```
mail_rex = "Value((?i)([a-z0-9!#$%&*+\\'=?^_`{|.}-~]+@(?:[a-z0-9](?:[a-z0-9]*[a-z0-9])?\.\.)+[a-z0-9](?:[a-z0-9]*[a-z0-9])?)")
zip_rex = 'Value((?i)\b\d{5}(?:[-\s]\d{4})?\b)'
phone_rex='Value(^(?:9[1-36][0-9]2[12][0-9]2[35][1-689]24[1-59]26[1-35689]27[1-9]28[1-69]29[1256])[0-9]{6}$)'
```

2.2.5.4.2 PII Categories

PrivacyNET supports a set of different categories for identifying PII, below is a table with a few samples of the different categories and rules fields that can be identified.

<i>FINANCIAL</i>	<i>PERSONAL</i>	<i>NATIONAL</i>	<i>TECH</i>	<i>OTHER</i>
<i>BANK</i>	<i>NAME</i>	<i>PASSPORT</i>	<i>URL</i>	<i>GEO LOCATION</i>
<i>CREDIT CARD</i>	<i>ADDRESS</i>	<i>DRIVER ID</i>	<i>IP</i>	<i>DATE</i>
<i>CVV</i>	<i>PHONE</i>	<i>NIF</i>	<i>IPv6</i>	<i>TIME</i>
<i>EXPIRY DATE</i>	<i>EMAIL</i>	<i>SSN</i>	<i>MAC</i>	<i>CREDENTIALS</i>

Table 1 - PII Categories

The rules can be defined in a domain specific language, which we provide a few examples below:

Example 1)

Category: Financial

RuleName: card_number

Code: (?-mix:((?:(?:\d{4}[-]?)\d{4})\d{15,16}))(![\d]) && ValidCreditCard())

This rule specified that values must match a given regular expression and pass the ValidCreditCard validation algorithm.

Example 2)

Category: Financial

RuleName: bitcoin

Code: `/(?![a-km-zA-HJ-NP-Z0-9])[13][a-km-zA-HJ-NP-Z0-9]{26,33}(?![a-km-zA-HJ-NP-Z0-9])/`

In this second example we just look for a given regular expression that can match the Bitcoin well-known format.

2.2.5.4.3 PII DSL Commands

The following list of commands are supported on the domain specific language.

Command	Description
AND, OR, NOT	Boolean operations to combine multiple rules
Value	Match a regular expression against the value of a property
Column	Match a regular expression against the name of a property
In	Check a value is contained in a user defined list of values
WordsIn	Check the words of a property value (split by space) are contained in a user defined list of values
Type	Match a regular expression against the type of a property (only available on structured or semi-structured inputs)
SQLType	Match a regular expression against the database type of a property (only available on SQL inputs)

ValidNIB	Perform the Luhn's algorithm [6] to check the number is a valid NIB
VALIDNIF	Perform the Luhn's algorithm [6] to check the number is a valid NIF (Portuguese Tax ID)
VALIDCC	Perform the Luhn's algorithm [6] to check the number is a CreditCard. Also check the bank prefix
VALIDNISS	Perform the Luhn's algorithm [6] to check the number is a valid Social Security number
PTAddress	Validate with at least 49% certain that a given text represents a Portuguese address. Check against dataset crawled from the Portuguese post office and performs statistical text analysis to compute a confidence score.
ENAddress	Validate with at least 49% certain that a given text represents an English UK address.
MinDigits	Check a number has at least n digits
MaxDIGITS	Check a number has at most n digits
MinSIZE	Check the number of characters of a given text is at least n
MaxSIZE	Check the number of characters of a given text is at most n
MINWORDS	Check a text has at least n words

Table 2 - DSL Commands

2.2.5.5 Policy Engine

In CyberSANE it was internally agreed DSA would be defined by CNR's tool from WP6 and PrivacyNET would implement the required primitives to enforce compliance. Inside the policy engine the main feature provided by PrivacyNET is the ability to register the retention period and set call back functions for a user defined data source. Since forcing erasure is outside the scope of this work, we focus on keeping track of the many used data sources as well as the retention policies associated with any of them. So first and foremost, it's a CMDB

for data sources that can contain PII's and a notification mechanism through the means of HTTP call backs, or SQL queries to request a source to delete data that is other than the policy permits.

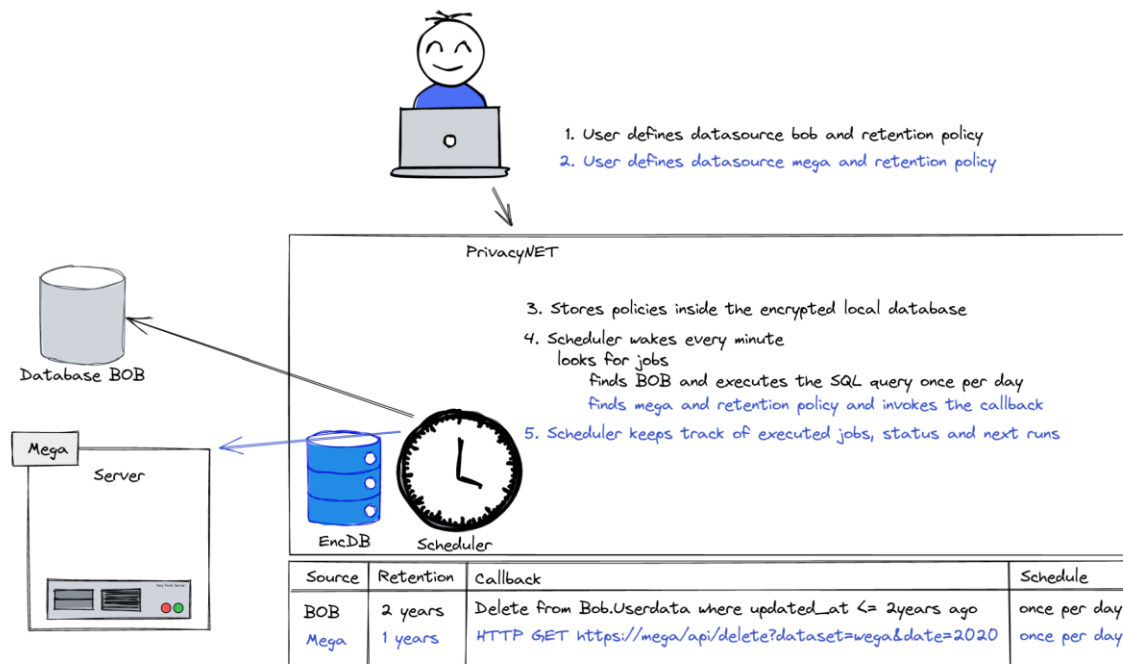


Figure 14 - Privacy Data Retention Engine

Both through local configurations and through HTTP API, users of PrivacyNET can save local policy rules inside the encrypted database. These policies are then frequently polled by the Scheduler looking for pending jobs. When such a job is detected, the job details are loaded, and callbacks are executed.

2.2.5.6 Homomorphic Functions

Homomorphic encryption is a large and complex field. In CyberSANE we focused on the partial homomorphic encryption techniques, that allow users to save data in encrypted storage and later search that storage without requiring full size downloads or decrypting all the stored information. Homomorphic search improves security and data privacy, but implies bigger latencies, higher resource consumption, slower queries, and limited functionalities. That's the price of privacy at the current state of the art.

Homomorphic search allows security aware users to encrypt sensitive data, before storing it on a third-party storage they don't control [7]. The hard part comes from ensuring the correct access level and weighing in the resource costs versus the security benefits. For a user to find the relevant content, it needs to download all content, decrypt it, compare it with the relevant search criteria and when the right resource is found, it then can search through

the clear text. This quickly becomes prohibitive both in terms of cost and time, making this approach infeasible for big data lakes [8].

To make the problem tractable researchers have proposed a generation of index table [9]. The index table keeps track of the words inside documents by creating a reverse lookup or inverted index. This process increases the computation resources when new documents are added to the encrypted storage but allow for quicker queries. In the traditional versions the indexes might require changes when new words are added to existing documents, we work around those limitation by resubmitting to the inverted index, documents that have been updated.

We avoid focusing on full homomorphic encryption theoretical discussions and implementation, to make a practical solution that ensure privacy aware storage.

Partial homomorphic encryption can be exercised based on a number of asymmetric encryption schemes. These include RSA, Goldwasser–Micali, Benaloh, ElGamal and Paillier and others [13].

In our approach we keep documents encrypted inside a key-value database. The documents are encrypted partial homomorphic encryption due to the nature of the task at hand. We follow largely the work layed out by CryptDB [14], with additional bloom filters and encrypted reverse document indexes to improve search performance on common use cases. Such as search by single word or list of words.

The implemented process to save document is as follows:

1. Get the list of unique words found in the text (hides the frequency)
2. Insert them into the bloom filter (speeds up negative word searches)
3. Encrypt each word with deterministic encryption
4. Obtain a hash blake2 or SHA256 of each encrypted word (hides the size of words)
5. Order the list randomly (hides the position in the text)
6. Document to be searched is encrypted with the RSA scheme and the list of hashes is attached.

Ignoring authentication, queries for text search based on multiple words, work following this process:

1. The client sends the words too lookup
2. The server checks the bloom filter to ensure words exist in the dataset
3. The server encrypts and hashes the words to be searched
4. The server searches for these hashes in the list and returns the document link if there is a match
5. The client asks the server to decrypt the document link and downloads the clear text document

There are 3 main operations to support homo search:

- 1) Save a new document
 - a. Store the document inside a Key Value database with key being the ID and Value the encrypted document with an encryption key.
- 2) Delete a document
 - a. Ask the encrypted storage to delete a document with key.
- 3) Search for a document based on words inside

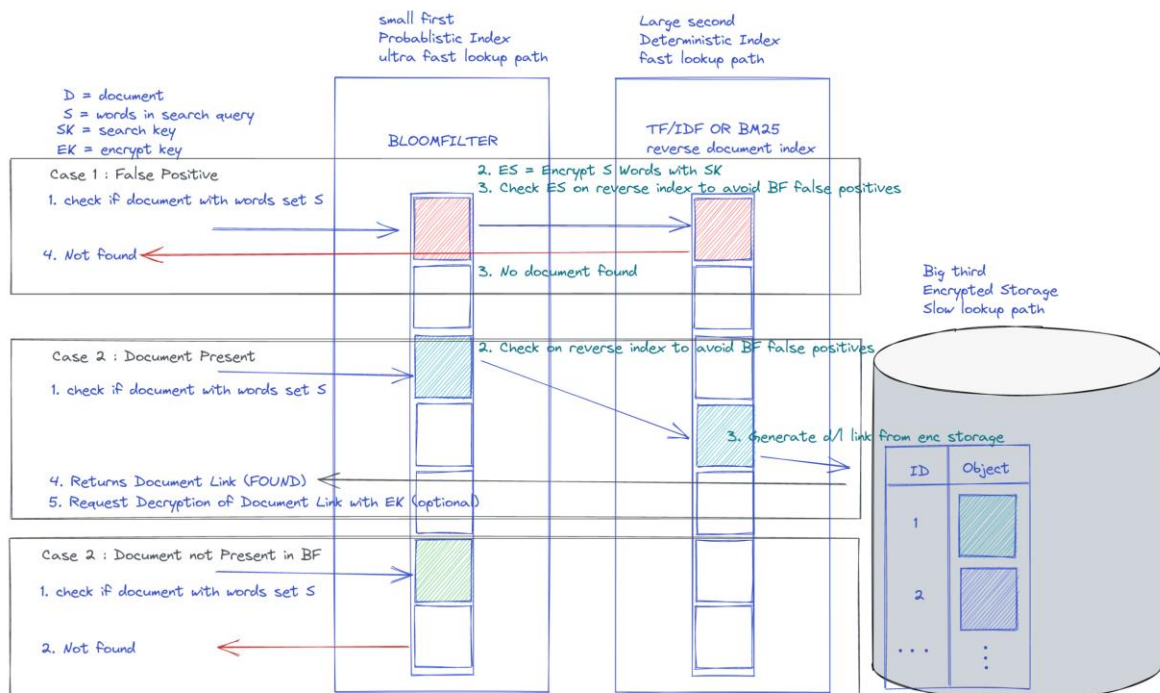


Figure 15 – Homosearch

2.2.5.7 Data Anonymization

PrivacyNET provides a data anonymization backend that has a hand-written parser and lexer that creates an Abstract Syntax Tree (AST) for the PAL (privacy anonymization language). That AST is processed through a tree walker interpreter which we call the runtime for PAL. Message passing between AST nodes is performed through shared memory and all nodes follow the same specifications for accessing data and creating new data. To ease the usage of the domain specific language, a web frontend with a single page application is developed to help operators to design the workflows visually and then export the rules in CAL to a standard format that can be passed into the backend runtime.

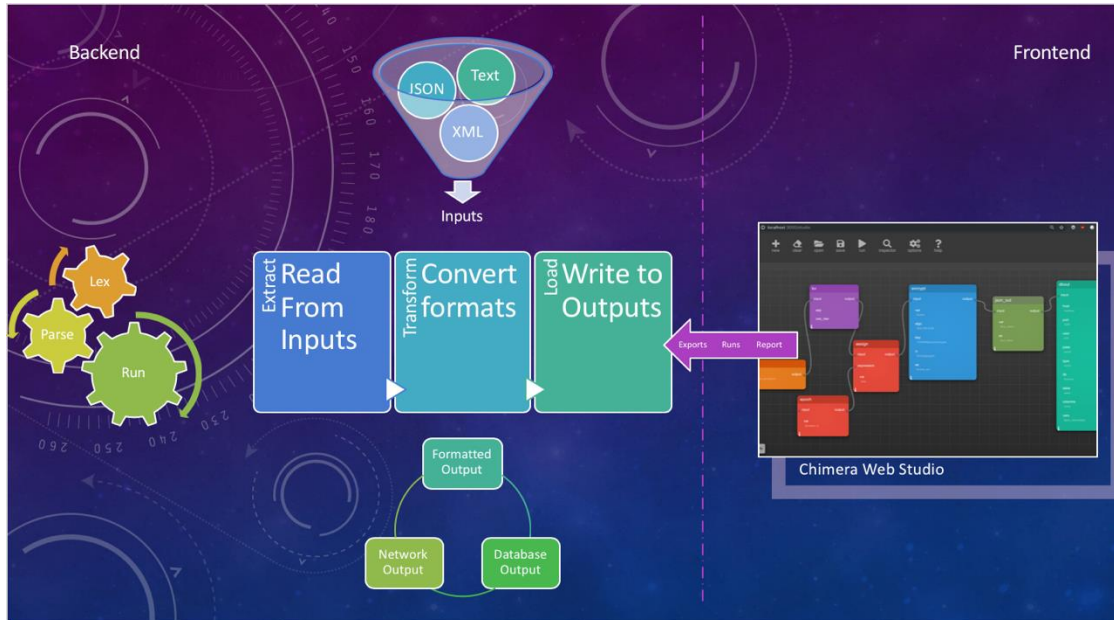


Figure 16 - Chimera Web Studio

The anonymisation module provides an API that is able to remotely transform data, by either performing encryption/decryption or by applying anonymisation techniques such as one-way hashing combined with k-Anonymity [10]/ l-Diversity [11]/ t-Closeness [12]. These techniques are realized through the implementation of generalization, masking and tokenization algorithms, combined with statistical combinatory analysis.

When applying anonymization through ABE, we need to consider the set of features present in the information being processed. Generically we can split data features into three categories: identifiers, quasi-identifiers and data. Where identifiers are key values that link to an individual directly. Quasi-identifiers don't link directly to an individual but can through combination with other quasi-identifiers can lead to individual identification.

The techniques leave the remaining data unchanged, and it can be treated independently from individuals, reducing the risk of identification to negligible levels. The data would then be virtually indistinguishable from randomly generated datasets.

The standalone GUI is based on the PDMFC's chimera tool, where a workflow for ABE (attribute-based encryption) can be leveraged to simplify setting up the required anonymization rules.

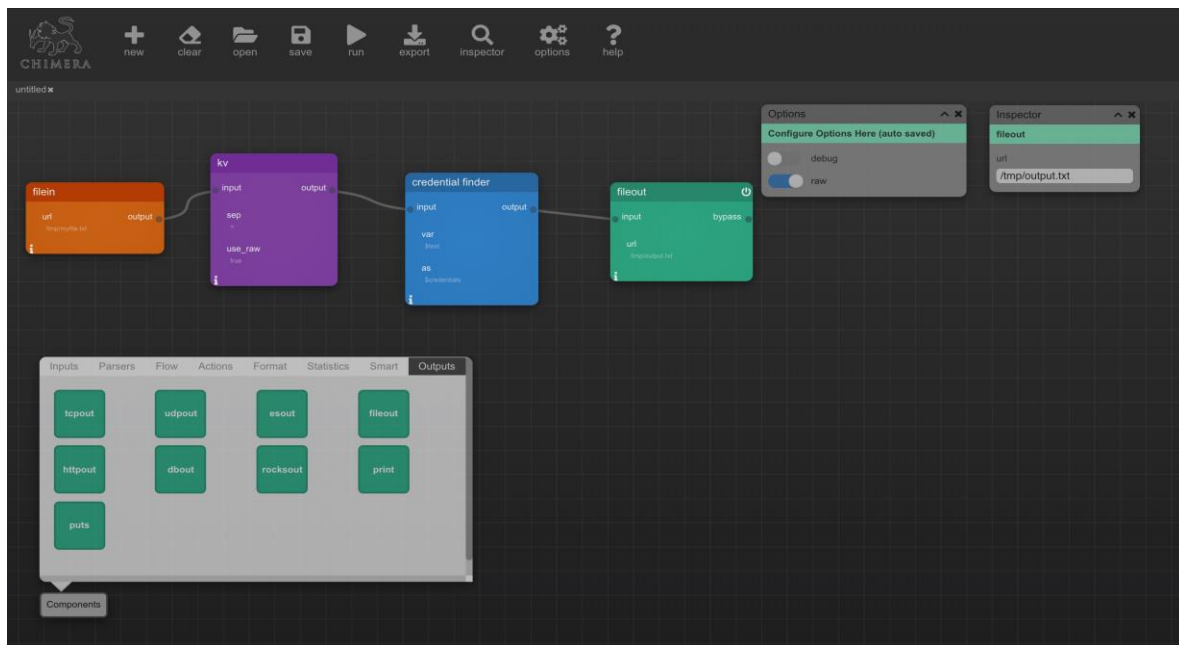


Figure 17 - GUI for rules creation

Regarding anonymization we support the following techniques:

- IP Masking
- Location Generalization (Local -> Region -> Country -> Continent)
- Geo Location Generalization (Reducing the decimal precision)
- Tokenization (replacement with pre-defined list values)
- Masking (replacing part of the content)
- Suppression

3 PrivacyNET & CyberSANE Integration

In this section, we shall go through the different services provides by PrivacyNET for consumption inside CyberSANE, either by the core platform or by other CyberSANE applications.

Visually the main application flow as defined on D2.4 is as follows:

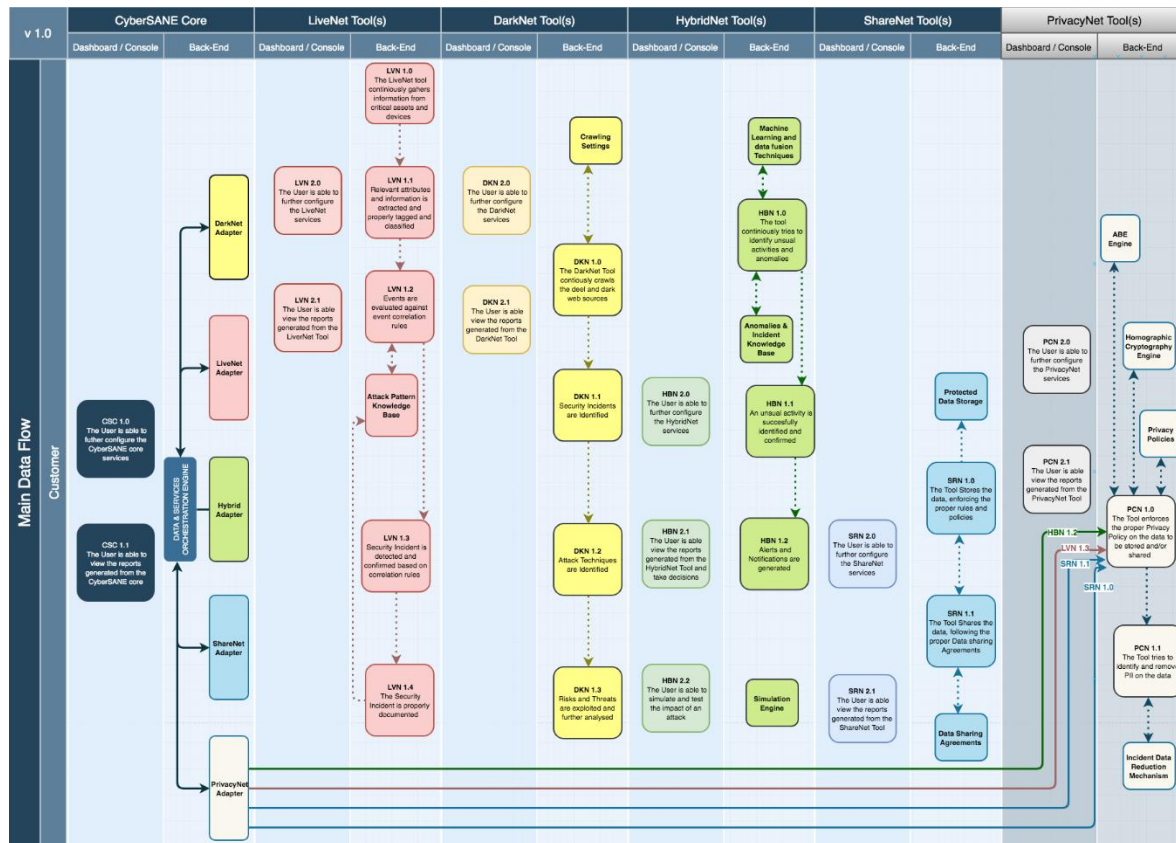


Figure 18 - PrivacyNET workflow inside CyberSANE

From here we can see the principal integration with PrivacyNET are through the features PCN 1.0, 1.1, 2.0 and 2.1, and the extended functionalities available.

For 1.0 and 1.1 functionalities are made available through the OpenAPI presented in 3.1.1. For 2.0 we provide Chimera Studio as shown on Figure 17 - GUI for rules creation. For 2.1 we provide a set of reports that can be exported to excel. As an example, we run the PII detector against a test database and generated a report on the number of PII detected using the methods presented in page 38.

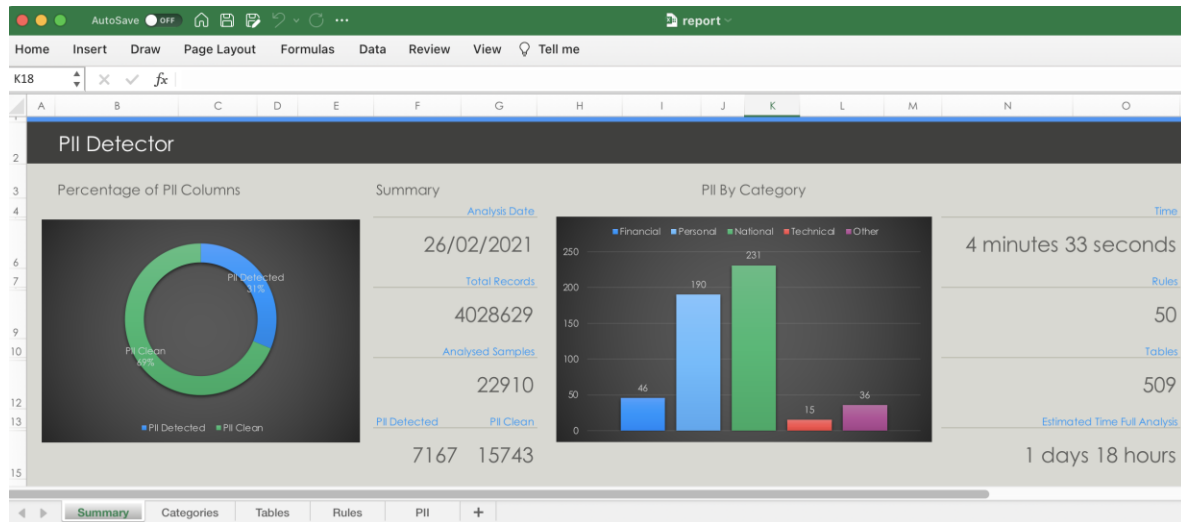


Figure 19 - Summary of PII report of a SQL database

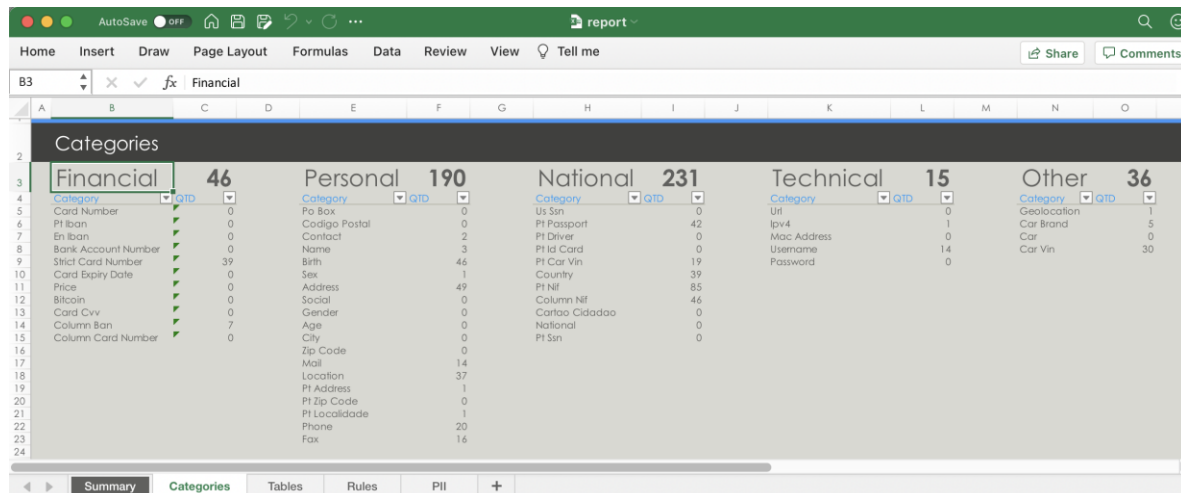


Figure 20 - Example of report from PII detection by category

3.1.1 OpenAPI Features

The full PrivacyNET OpenAPI specification is around 1000 lines, which make it too big to reproduce in this document, it is stored in the project git repository and has become a live document as the API might be improved or changed going forward. We resume this section by mapping the features specified on deliverable D2.4 with the web endpoints and description for each API call.

3.1.1.1 PRI-F-010.1 Encrypt Data

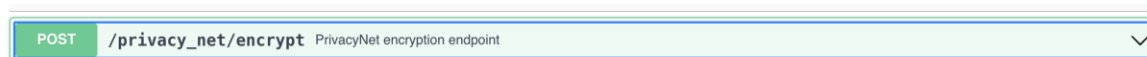


Figure 21 - Encrypt Endpoint

3.1.1.2 PRI-F-010.2 Decrypt data

POST `/privacy_net/decrypt` PrivacyNet decryption endpoint

3.1.1.3 PRI-F-020.1 Anonymization of security incident data

POST `/privacy_net/anonymize_netflow_raw` PrivacyNet endpoint for raw netflow anonymization, masks ips with AES-FFX

POST `/privacy_net/anonymize_netflow` PrivacyNet endpoint for raw netflow anonymization, masks ips with AES-FFX

3.1.1.4 PRI-F-020.2 Anonymization of security incident reports

POST `/privacy_net/anonymize_report` PrivacyNet endpoint for security incident report anonymization, masks the concepts selected by the api user

3.1.1.5 PRI-F-020.3 Dynamic Data Masking

POST `/privacy_net/data_masking` Provide a service to encrypt data dynamically, based on the options specified on inputs. For some types of reports, encrypting or hashing to anonymize takes all value out of the report. This service would allow to specify a masking pattern and apply a simple string replacement only for the required fields.

3.1.1.6 PRI-F-020.4 Map & Merge Fields

POST `/privacy_net/data_masking` Provide a service to encrypt data dynamically, based on the options specified on inputs. For some types of reports, encrypting or hashing to anonymize takes all value out of the report. This service would allow to specify a masking pattern and apply a simple string replacement only for the required fields.

3.1.1.7 PRI-F-020.5 Filter

POST `/privacy_net/filter_merge` When doing anonymization of data, sometimes a few entities stand out for either appearing too much or more trickily too little, to avoid leaking statistical information it's necessary to either apply generalization, field merging or mapping the data to another format. This service will allow for dynamically transformation of attributes to ensure k-anonymity is possible. This service allows for incident data to be filtered either by column (attribute) or by row. This allows for data subsetting while ensuring referential integrity.

3.1.1.8 PRI-F-020.6 Validation

POST `/privacy_net/validation` Provide a service to create cryptographic hashes of input data, based on the options provided by the consuming components.

3.1.1.9 PRI-F-030.1 Data Encryption

POST `/privacy_net/homo_encryption` Provide a service to encrypt data with homomorphic primitives.


3.1.1.10 PRI-F-030.2 Data Decryption

POST `/privacy_net/homo_decryption` Provide a service to decrypt data with homomorphic primitives.


3.1.1.11 PRI-F-030.3 Transformation

This operation was scrapped, and no API was provided, has it didn't fit any mandatory use case.


3.1.1.12 PRI-F-030.4 Search

POST `/privacy_net/homo_search` Provide a service to search for strings / words on previously encrypted data. 


3.1.1.13 PRI-F-040.1 PII Detection

POST `/privacy_net/pii_detection` Provide a service that attempts to detect PII inside of structure and unstructured data. The following are categories and respective examples of the types of data that should be auto discovered by this service 


3.1.1.14 PRI-F-040.2 PII Redaction / Privacy Rules Workflow Engine

POST `/privacy_net/pii_scrubbing` Provide a service that receives a list of anonymization rules and applies them on the input data. This is a lower level API that can be leveraged by other APIs to produce anonymized reports. The rules will be based on the anonymization models that will come out of WP7. This will be the service responsible to parsing, instantiating the respective computational functions and apply them to the source data. 


3.1.1.15 PRI-F-040.3 Privacy Rules Operation Metrics

POST `/privacy_net/metrics` Provide a service that receives lists the operation KPIs recorded for the usage of the PrivacyNet services. This will provide important metrics such as number of executed workflows, number of anonymization formats supported, number of workflow templates, average execution times, p95/p99 metrics. 


3.1.1.16 PRI-F-050.1 Data Access Management

POST `/privacy_net/data_access_management` Manage the complete lifecycle of digital identities and access roles in one system and enables an organization to classify any access or profile with a level of criticality and to label every right of access PII data. This endpoint in particular can save new datasources and their respective categories and permissions. 


3.1.1.17 PRI-F-050.2 Save Data Retention

POST `/privacy_net/data_retetion` Set the data retention period for a given data asset. The retention policy is enforced by calling the input callback procedure on the system the data is originally from. The CRON frequency will be used to define how often the retention period verification should occur. 


3.1.1.18 PRI-F-050.3 Retrieve Data Retention

GET `/privacy_net/data_retetion/{id}` Retrieve the retention policy applied to a given data asset. 

3.1.1.19 PRI-F-050.4 Register PII Data Processing

POST `/privacy_net/pii_registration` RRegisters that a given component required the specified Data Asset for processing. 

3.1.1.20 PRI-F-050.5 Retrieve PII Data Processing Details

GET `/privacy_net/pii_registration/{id}` Retrieves the details for a given data asset ID. 

3.1.1.21 PRI-F-050.6 Notify PII Data Usage

POST `/privacy_net/pii_notification/{id}` Registers that a given component has processed a given PII data asset

3.1.1.22 PRI-F-050.7 Retrieve PII Data Processing History

POST `/privacy_net/pii_usage/{id}` Retrieves the list of usages recorded since a data asset was original created on the Privacy Net system.

3.1.2 Custom pipeline Examples

Noteworthy and exceptional inside PrivacyNet is the ability to create custom pipelines for data processing, where all rules can be combined dynamically. Sacrificing performance for flexibility. As such examples are abundant we present 4 examples that were used in the integration with ShareNet to demonstrate the capabilities, and clarify how the tool can be useful.

3.1.2.1 Anonymization of lessons learned

This example presents the web endpoint configured to anonymize destination ips through an ipmask generalization. It will remove the least significant bits from the IP address by applying a netmask of 255.255.0.0. Consider the following configuration (only the relevant bits pertaining to the operation in question) as presented below:

Rules.toml

```
# JGET
# Since: 1.1.49
# Retrieve a value from a JSON field that is arbitrarily deep
[jget.ips]
# Field source to retrieve json from, can be the special value _raw to retrieve from rb.Bytes or anything else to
# retrieve from each event
Field="_raw"
# Field where to store the result
# if Field=RawField then we will retrieve set value on rb.Meta[$Dest] else on rb.Events[i][$Dest]
Dest="ips"
# Expression to set
Path="anomalies.#.destinationIp"

# JSET
# Since: 1.1.49
# Retrieve a value from a JSON field that is arbitrarily deep
[jset.dst_ips]
# Field source to retrieve json from, can be the special value _raw to retrieve from rb.Bytes or anything else to
# retrieve from each event
Field="_raw"
# Field where to store the result
# if Field=RawField then we will retrieve set value on rb.Meta[$Dest] else on rb.Events[i][$Dest]
Value="ips"
# Expression to set
Path="anomalies.#.destinationIp"
```

```
#### ipmask ###
# Apply a netmask to an IP address as means to anonymize the sensitive lower octets
# ipmask("192.168.0.1","255.255.0.0") -> "192.168.0.0"
[ipmask.ary]
# List of fields to perform ipmasking on
Fields=["__MVVALUE__"]
# Subnet to apply the ipmasking
Mask="255.255.0.0"
[mvmap.ips]
Field="ips"
Rule="ipmask.ary"
```

Config.toml

```
[http.generic]
Path="/lessons_learned/mask_destip"
Rules=["filein.#infile",
      "jget.ips",
      "mvmap.ips",
      "jset.dst_ips",
      "fileout.#outfile"]
```

And the following input (lessons_learned.json):

```
{
  "id": 1,
  "name": "SQL Injections - Input Validation",
  "description": "Input validation\n\nThe validation process is aimed at verifying whether or not the type of input submitted by a user is allowed. Input validation makes sure it is the accepted type, length, format, and so on. Only the value which passes the validation can be processed. It helps counteract any commands inserted in the input string. In a way, it is similar to looking to see who is knocking before opening the door.\n\nValidation shouldn't only be applied to fields that allow users to type in input, meaning you should also take care of the following situations in equal measure:\n\n  Use regular expressions as whitelists for structured data (such as name, age, income, survey response, zip code) to ensure strong input validation.\n  In case of a fixed set of values (such as drop-down list, radio button), determine which value is returned. The input data should match one of the offered options exactly.",
  "assets": [
    {
      "lessonAssetId": 5,
      "id": 25,
      "name": "MariaDB"
    },
    {
      "lessonAssetId": 3,
      "id": 34,
      "name": "Oracle DB"
    },
    {
      "lessonAssetId": 4,
      "id": 26,
```

```
"name": "PostgreSQL"
},
],
"controls": [],
"threats": [
  {
    "lessonThreatId": 1,
    "id": 10,
    "name": "SQL Injection"
  }
],
"vulnerabilities": [
  {
    "lessonVulnerabilityId": 2,
    "id": "CVE-1999-0001"
  }
],
"attackPatterns": [
  {
    "id": 11,
    "timestamp": "2021-04-06 23:06:42",
    "observerProduct": "Encrypted Network Intrusion Detection",
    "observerVendor": "FORTH",
    "ruleId": "0",
    "ruleDescription": "Hydra (Web server login attempt)",
    "ecsType": "ATTACK_PATTERN"
  }
],
"securityIncidents": [
  {
    "id": 16,
    "timestamp": "2021-04-06 23:16:37",
    "destinationIp": "192.168.56.103",
    "destinationPort": "443",
    "eventSeverity": "Unknown",
    "networkTransport": "tcp",
    "observerProduct": "Encrypted Network Intrusion Detection",
    "observerVendor": "FORTH",
    "ruleId": "5",
    "ruleDescription": "Metasploit (File/Directory scanning to web server in victim machine)",
    "sourceIp": "192.168.56.101",
    "sourcePort": "41367",
    "ecsType": "SECURITY_INCIDENT"
  }
],
"anomalies": [
  {
    "id": 24,
    "timestamp": "2021-04-10T11:20:51.000Z",
    "destinationIp": "109.99.165.100",
    "destinationPort": "5058",
    "eventAction": "",
    "eventCategory": "Reconnaissance",
    "eventEnd": ""
  }
]
```



```
"eventId": "-F3GH45B1aLxeSxNRfTt",
"eventOutcome": "Unknown",
"eventSeverity": "Unknown",
"eventTimezone": "UTC",
"eventType": "Port Scanning",
"fileName": "alerts.json",
"filePath": "/var/sensor/logs/alerts/",
"hostName": "94d0cb529d714f4389be746117ba4553",
"hostIp": "158.211.53.52",
"hostMac": "",
"message": "This is a demo message",
"networkTransport": "ftp",
"observerIp": "158.211.53.52",
"observerName": "ML Sensor",
"observerType": "sensor",
"observerProduct": "SiVi",
"observerVendor": "Sidroco",
"organizationId": "KN",
"ruleDescription": "Machine learning rule",
"sourceHostname": "snf-16552",
"sourceIp": "251.134.130.21",
"sourceMac": "52:54:00:f8:21:4b",
"sourcePort": "5200",
"userName": "aut",
"ecsType": "ANOMALY"
},
{
  "id": 25,
  "timestamp": "2021-04-10T11:20:51.000Z",
  "destinationIp": "109.99.165.101",
  "destinationPort": "5058",
  "eventAction": "",
  "eventCategory": "Reconnaissance",
  "eventEnd": "",
  "eventId": "-F3GH45B1aLxeSxNRfTt",
  "eventOutcome": "Unknown",
  "eventSeverity": "Unknown",
  "eventTimezone": "UTC",
  "eventType": "Port Scanning",
  "fileName": "alerts.json",
  "filePath": "/var/sensor/logs/alerts/",
  "hostName": "94d0cb529d714f4389be746117ba4553",
  "hostIp": "158.211.53.52",
  "hostMac": "",
  "message": "This is a demo message",
  "networkTransport": "ftp",
  "observerIp": "158.211.53.52",
  "observerName": "ML Sensor",
  "observerType": "sensor",
  "observerProduct": "SiVi",
  "observerVendor": "Sidroco",
  "organizationId": "KN",
  "ruleDescription": "Machine learning rule",
  "sourceHostname": "snf-16552",
```

```
"sourceIp": "251.134.130.21",
"sourceMac": "52:54:00:f8:21:4b",
"sourcePort": "5200",
"userName": "aut",
"ecsType": "ANOMALY"
}
]
```

Would produce the following output (reduced to show the relevant differences):

```
{
  "id": 1,
  ...
  "anomalies": [
    {
      "id": 24,
      "timestamp": "2021-04-10T11:20:51.000Z",
      "destinationIp": "109.99.0.0",
      ...
    },
    {
      "id": 25,
      "timestamp": "2021-04-10T11:20:51.000Z",
      "destinationIp": "109.99.0.0",
      ...
    }
  ]
}
```

The previous example might be too long to be understood easily as first, so let us break it down into the most relevant changes and present it with reproduceable linux commands.

Assumptions:

1. Lessons Learned in STIX 2.0 JSON format at path test/data/lessonlearned.json
2. Cat [15], jq[16], curl[17] system tools installed
3. PrivacyNET with the configuration presented above running at localhost:8080

```
cat test/data/lessonslearned.json | jq -c '.anomalies[] | {id,destinationIp}'
```

Source

```
{ "id": 24, "destinationIp": "109.99.165.100" }
{ "id": 25, "destinationIp": "109.99.165.101" }
```

Testing with Curl

```
curl -s -X POST localhost:8080/api/lessons_learned/mask_destip \
-H "Content-Type: application/json" \
-d '{"infile":"test/data/lessonslearned.json","outfile":"lout.json"}' \
| jq -c '.anomalies[] | {id,destinationIp}'
```

Output

```
{"id":24,"destinationIp":"109.99.0.0"}
{"id":25,"destinationIp":"109.99.0.0"}
```

This reduction of input a output through usage of the excellent jq tool, allows for quick visual inspection of changes. There a lot to unpack here, lets start at the top with http.generic input which by convention defines a new http rule inside an http router, that maps a path to a pipeline. The pipeline rules must be either present inside the rules.toml file or be simple enough to be defined implicitly. The rules for implicitly defining rules are complex, rule type dependent and out of scope for this document. We leave the reader with the general intuition about implicit rule definition, the `<rule_type>.<rule_name>([<mandatory args>])`. Rule name is often used as input field and output field.

Next the rules defined would follow the dataflow:

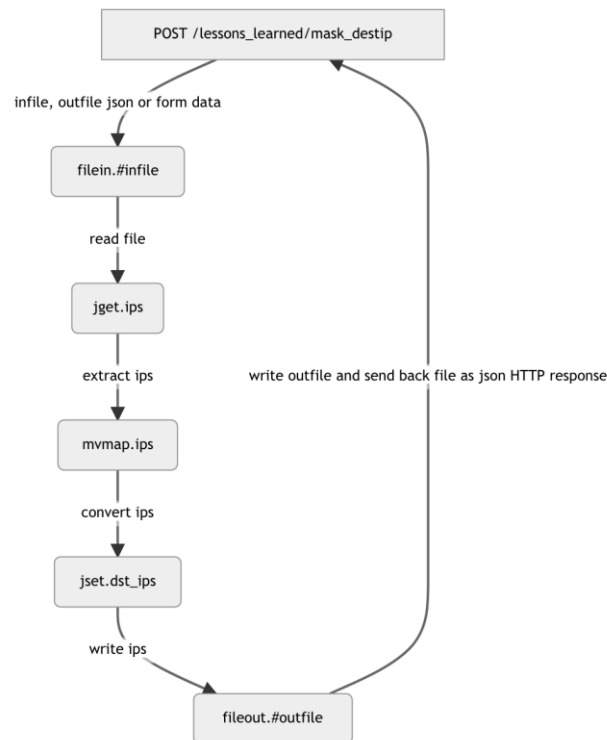


Figure 22 - Http API to scrub lessons learned

3.1.2.2 Anonymization of assets with inline rules

Pipeline

```
[http.assets]
Path="/lessons_learned/mask_assets"
Rules=["filein.#infile",
      "jget.ips(assets.#.name)",
      "mvmmap.ips(mask.__VALUE__(###))",
      "jset.ips(assets.#.name)",
      "fileout.#outfile"]
```

Command

```
cat test/data/lessonslearned.json | jq -c '.assets[]'
```

Source

```
{"lessonAssetId":5,"id":25,"name":"MariaDB"}
{"lessonAssetId":3,"id":34,"name":"Oracle DB"}
{"lessonAssetId":4,"id":26,"name":"PostgreSQL"}
```

Testing with Curl

```
curl -s -X POST localhost:8080/api/lessons_learned/mask_assets \
-H "Content-Type: application/json" \
-d '{"infile":"test/data/lessonslearned.json","outfile":"lout.json"}' \
| jq -c '.assets[]'
```

Output

```
{"lessonAssetId":5,"id":25,"name":"__iaDB"}
{"lessonAssetId":3,"id":34,"name":"__cle DB"}
{"lessonAssetId":4,"id":26,"name":"__tgreSQL"}
```

Data flow

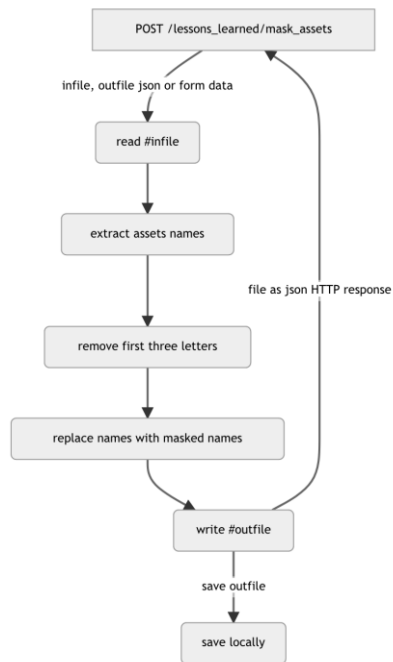


Figure 23 - Dataflow mask assets

3.1.2.3 Anonymization of incidents

Pipeline

```

[http.incidents]
Path="/lessons_learned/mask_incidents"
Rules=["filein.#infile",
      "jget.dips(securityIncidents.#.destinationIp)",
      "jget.sips(securityIncidents.#.sourceIp)",
      "mvmmap.dips(ipmask.dips(255.0.0.0))",
      "mvmmap.sips(ipmask.sips(255.0.0.0))",
      "jset.dips(securityIncidents.#.destinationIp)",
      "jset.sips(securityIncidents.#.sourceIp)",
      "fileout.#outfile"]
  
```

Command

```
cat test/data/lessonslearned.json | jq -c '.securityIncidents[]'
```

Source

```
{
  "id": 16,
  "timestamp": "2021-04-06 23:16:37",
  "destinationIp": "192.168.56.103",
  "destinationPort": "443",
  "eventSeverity": "Unknown",
  "networkTransport": "tcp",
  "observerProduct": "Encrypted Network Intrusion Detection",
  "observerVendor": "FORTH",
  "ruleId": "5",
  "ruleDescription": "Metasploit (File/Directory scanning to web server in victim machine)",
  "sourceIp": "192.168.56.101",
  "sourcePort": "41367",
  "ecsType": "SECURITY_INCIDENT"
}
```

Testing with Curl

```
curl -s -X POST localhost:8080/api/lessons_learned/mask_incidents \
-H "Content-Type: application/json" \
-d '{"infile":"test/data/lessonslearned.json","outfile":"lout3.json"}' \
| jq '.securityIncidents[] | {id, destinationIp, sourceIp}'
```

Output

```
{
  "id": 16,
  "destinationIp": "192.0.0.0",
  "sourceIp": "192.0.0.0"
}
```

Dataflow

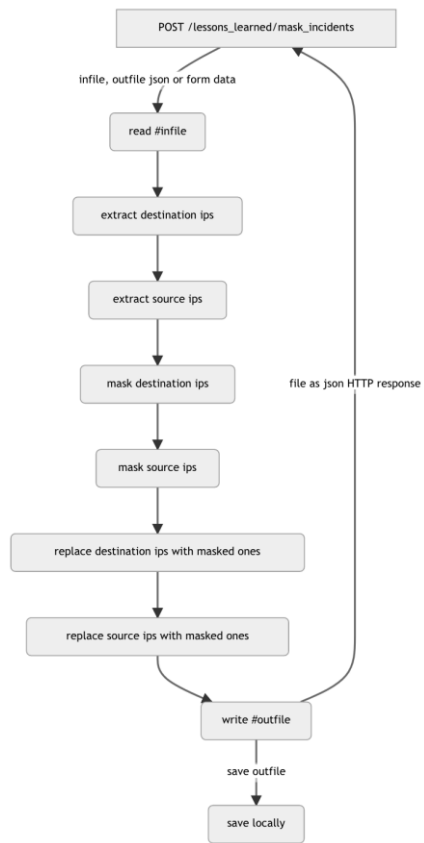


Figure 24 - Incident anonymization dataflow

3.1.2.4 Anonymization of Anomalies

Pipeline

```
[http.anomalies]
Path="/lessons_learned/mask_anomalies"
Rules=["filein.#infile",
"jget.dips(anomalies.#.destinationIp)",
"jget.sips(anomalies.#.sourceIp)",
"jget.hips(anomalies.#.hostIp)",
"jget.oips(anomalies.#.observerIp)",
"jget.hmac(anomalies.#.hostMac)",
"jget.smac(anomalies.#.sourceMac)",
"mvmmap.dips(ipmask.dips(255.0.0.0))",
"mvmmap.sips(ipmask.sips(255.0.0.0))",
"mvmmap.hips(ipmask.hips(255.0.0.0))",
"mvmmap.oips(ipmask.oips(255.0.0.0))",
"mvmmap.hmac(mask.hmac(__:__:##:##:##:##))",
"mvmmap.smac(mask.smac(__:__:##:##:##:##))",
"jset.dips(anomalies.#.destinationIp)",
"jset.sips(anomalies.#.sourceIp)",
"jset.hips(anomalies.#.hostIp)",
"jset.oips(anomalies.#.observerIp)",
"jset.hmac(anomalies.#.hostMac)",
"jset.smac(anomalies.#.sourceMac)",
"fileout.#outfile"]
```

Command

```
cat test/data/lessonslearned.json |
jq '.anomalies[] | {id,destinationIp, sourceIp, hostIp, observerIp, hostMac, sourceMac}'
```

Source

```
{
  "id": 24,
  "destinationIp": "109.99.165.100",
  "sourceIp": "251.134.130.21",
  "hostIp": "158.211.53.52",
  "observerIp": "158.211.53.52",
  "hostMac": "",
  "sourceMac": "52:54:00:f8:21:4b"
}
{
  "id": 25,
  "destinationIp": "109.99.165.101",
  "sourceIp": "251.134.130.21",
  "hostIp": "158.211.53.52",
  "observerIp": "158.211.53.52",
  "hostMac": "",
  "sourceMac": "52:54:00:f8:21:4b"
}
```


Testing with Curl

```
curl -s -X POST localhost:8080/api/lessons_learned/mask_anomalies \
  -H "Content-Type: application/json" \
  -d '{"infile":"test/data/lessonslearned.json","outfile":"lout3.json"}' \
  | jq '.anomalies[] | {id,destinationIp, sourceIp, hostIp, observerIp, hostMac, sourceMac}'
```

Output

```
{
  "id": 24,
  "destinationIp": "109.0.0.0",
  "sourceIp": "251.0.0.0",
  "hostIp": "158.0.0.0",
  "observerIp": "158.0.0.0",
  "hostMac": "",
  "sourceMac": "__:__:00:f8:21:4b"
}
{
  "id": 25,
  "destinationIp": "109.0.0.0",
  "sourceIp": "251.0.0.0",
  "hostIp": "158.0.0.0",
  "observerIp": "158.0.0.0",
  "hostMac": "",
  "sourceMac": "__:__:00:f8:21:4b"
}
```

Dataflow

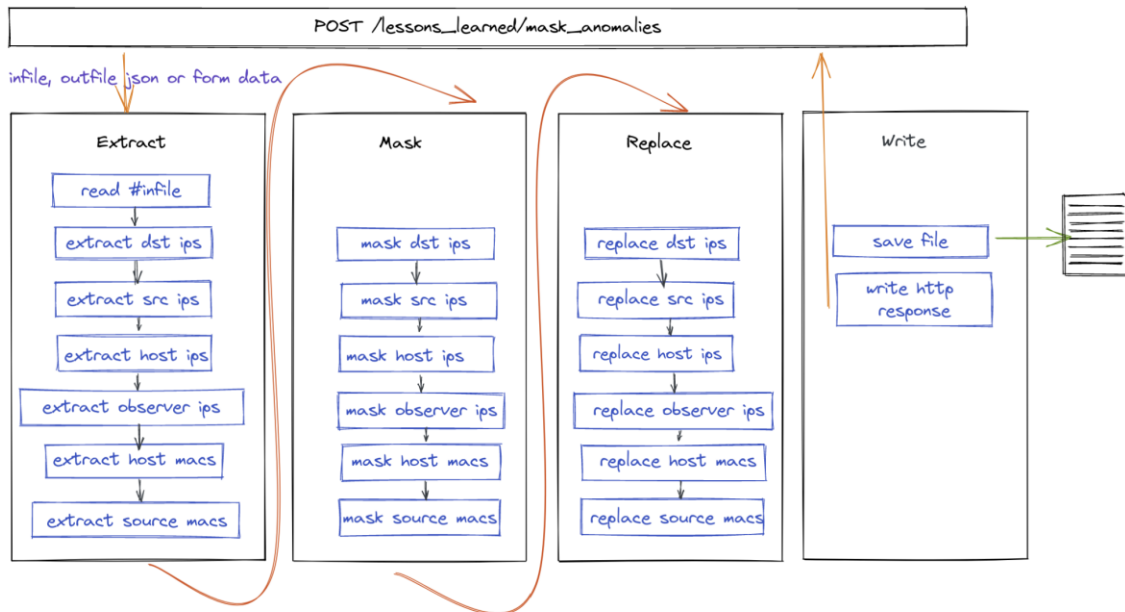


Figure 25 - Mask Anomalies Dataflow

4 Conclusions and Future Directions

The PrivacyNET provides the building blocks for anonymization, encryption, homomorphic search, and privacy policy enforcement inside CyberSANE. The PrivacyNET coordinates with the components of the CyberSANE system to ensure desired-levels of data protection for sensitive incident-related information in the context of the envisaged pilots.

The main purpose is to manage and orchestrate the application of the innovative privacy mechanisms and maximize achievable levels of confidentiality and data protection towards compliance with the highly demanding provisions of the GDPR in the context of protecting sensitive incident-related information within and outside CIs. The orchestration approach of the CyberSANE allows applying the most appropriate security and data protection methods depending on the user's privacy requirements, which cover a wide range of techniques including anonymization, location privacy, obfuscation, pseudonymization, searchable encryption.

The following table summarizes the feature set:

Functionalities	Grouping	Services
Anonymization & Encryption	Format Preserving ABE	Encryption of resting data
		Decryption of resting data
	Incident Data Redaction	Anonymization of incident data
		Anonymization of incident reports
		Dynamic Data masking
		Map & Merge Fields
		Filter & Validate
Homomorphic Encryption	Privacy Preserving Transformations	Privacy Preserving Computation / Transformation
		Privacy Preserving Storage / Data encryption / Data Decryption
		Privacy Preserving Search

Privacy Policy Enforcement	Security Incident Data Redaction	PII Detection
		PII Redaction / Privacy Rules Workflow
		Privacy Rules Operation Metrics
	Privacy Policy Enforcement	Data Access Management
		Set Data Retention
		Get Data Retention
		Register PII Data Processing
		Retrieve PII Data Processing Details
		Notify PII Data Usage
		Retrieve PII Data Processing History

Table 3 - PrivacyNet Services' Grouping and Mapping with Component's functionalities

PrivacyNET supports the most common formats such as JSON and XML, allowing for user defined taxonomies for data structures. By providing the lower-level interchangeable actions that can be dynamically defined by users.

As future work we identified a set of improvements that could use the adoptability of the PrivacyNET by third parties. Namely increasing the number of source connectors for dominant data players in the current market, for instance the ability to input data directly from Google Sheets through only an URL and respective access token, or to acquire data directly from Salesforce APIs, or any other frequently used third party system with a track record for storing PII and sensitive data. On the opposite side and following the same rational, it makes sense to extend the list of sink connectors to where data can be exported to.

Regarding the dynamic nature of processing required, the current anonymization language is computed through a tree-walking interpreter, which is simpler and quicker to implement but leaves some performance on the table for tight loop, or high-pressure points. Extending the interpreter to include a JIT and perform dynamic node replacement directly on the ASTs would be a worthy endeavour for supporting terabyte and larger datasets. Continuing on the language front more features could extend its applicability, support for window functions, generic aggregation functions and user defined functions could make the language able to

tackle new domains. Albeit at the cost of extra complexity, research into how to keep the language accessible to new users while retaining its core features, should be fertile ground for innovation.

In terms of UX improvement, for demos and quick onboarding of new users, having a set of default pipelines in place, that allows for users to immediately try and experiment with, without the risk of damaging the system, through a visual web GUI should in our opinion increase the visibility of the tool outside research projects.

As the core configurations are file based and can be persisted fully to disk, retaining a common know format, they are well suited to be stored in GIT. This led us to consider a module system that could support namespaces and import configurations directly from github.com or GitLab hosted repositories.

On the storage front, extending the current storage primitive from key value store with encryption support, with the right primitives to store system and audit logs efficiently, would fill a space in dire need for disruption. As the current open-source solutions are mostly focused on Elastic Search which is a poor fit for low resource environment. For such alternative to be possible, we envision a system that can make extensive use of BM25 or TF-IDF in conjunction with bloom filters and time-based buckets. Creating an engine that would be performant for time-based filtering and full text search queries.

Lastly, improving auditing capabilities for privacy rules, could improve visibility inside CyberSANE for potential misuse and provide forensic evidence for analysts to use.

5 References

- [1] Structured Threat Information Expression (STIX), Available online: <https://oasis-open.github.io/cti-documentation/stix/intro>
- [2] Communicating sequential processes (CSP), C. A. R. Hoare, Communications of the ACM Volume 21 Issue 8 Aug. 1978 pp 666–677 <https://doi.org/10.1145/359576.359585>
- [3] Portable Operating System Interface (POSIX), Available online: https://www.opengroup.org/austin/papers/posix_faq.html
- [4] Using NLP and Machine Learning to Detect Data Privacy Violations, Paulo Silva, Carolina Gonçalves, Carolina Godinho, Nuno Antunes, Marilia Curado, 2020, https://eg.uc.pt/bitstream/10316/95068/1/WORKSHOP_ON_SECURITY_AND_PRIVACY_IN_BIG_DATA_Camera_Ready.pdf
- [5] Domain Specific Languages, Martin Fowler, 2010, Addison-Wesley, <https://martinfowler.com/books/dsl.html>
- [6] Luhn's Algorithm, US patent US2950048A, Hans Peter Luhn, 1960, <https://worldwide.espacenet.com/patent/search/family/003449488/publication/US2950048A?q=pn%3DUS2950048A>
- [7] A survey of provably secure searchable encryption, Bösch, C., Hartel, P., Jonker, W. and Peter, A., 2014. ACM Computing Surveys (CSUR), 47(2), pp.1-51, <https://dl.acm.org/doi/10.1145/2636328>
- [8] Searchable encryption for healthcare clouds: a survey. Zhang, R., Xue, R. and Liu, L., 2017, IEEE Transactions on Services Computing, 11(6), pp.978-996 <https://ieeexplore.ieee.org/ielam/4629386/8567855/8066356-aam.pdf>
- [9] Inverted index based multi-keyword public-key searchable encryption with strong privacy guarantee, Wang, B., Song, W., Lou, W. and Hou, Y.T., 2015, IEEE Conference on Computer Communications pp. 2092-2100, https://www.cnsr.ictas.vt.edu/publication/Wang_2015_INFOCOM.pdf
- [10] k-Anonymity: A Model for Protecting Privacy, Latanya Sweeney, 2002, J. Uncertain. Fuzziness Knowl. Based Syst, pp 557-570, <http://cs.jhu.edu/~sdoshi/jhuisi650/papers/kanonymity.pdf>
- [11] l-Diversity: Privacy Beyond k-Anonymity, Machanavajjhala, A., Gehrke, J., Kifer, D., & Venkatasubramanian, M., 2006, ICDE, <http://www.cs.cornell.edu/people/dkifer/ldiversityTKDDdraft.pdf>
- [12] t-Closeness: Privacy Beyond k-Anonymity and l-Diversity, Li, N., Li, T., & Venkatasubramanian, S. (2007).. 2007 IEEE 23rd International Conference on Data Engineering, 106-115. http://www.cs.purdue.edu/homes/ninghui/papers/t_closeness_icde07.pdf

- [13] Partially Homomorphic Cryptosystems, Wikipedia, last access on 2021, https://en.wikipedia.org/wiki/Homomorphic_encryption#Partially_homomorphic_cryptosystems
- [14] CryptDB: Protecting confidentiality with encrypted query processing, R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan, 2011, Proceedings of the 23rd ACM Symposium on Operating Systems Principles, pages 85–100 https://web.cs.ucdavis.edu/~franklin/ecs228/2013/popa_etal_sosp_2011.pdf
- [15] Cat – concatenate files and print on standard output, gnu tool, last accessed on 29/Jan/2022 at <https://man7.org/linux/man-pages/man1/cat.1.html>
- [16] Jq – Json filter tool, cli tool, last accessed on 29/Jan/2022 at <https://stedolan.github.io/jq/manual/>
- [17] Curl - command line tool and library for transferring data with URLs, 1998, last assessed on 29/Jan/2022 at <https://curl.se/>

Annex I – Config Spec

```
## FILE ##
# Stanza example for monitoring files
[file.csv]
# The filepath to monitor, can be absolute or relative to work dir
Path="in.csv"
# Array with the names of the rules to be applied to this input
Rules=["rex.csv","ff1.dstip","hash.srcip","recode.srcip(Extended,EmailAlpha)"]
# Array of outputs where processed events will be sent after they go through the pipeline
# IMPORTANT: At the moment this array can only have 1 output, having 2 or more will cause random lockups
Outputs=["kv"]
# File encoding, by default utf-8 if nothing else selected
# Full list available at docs/encode_formats.md
Encoding="utf-8"
# Don't keep track in the internal db of the current offset that's already processed
NoTracking=false
# Don't monitor file forever, just read once until EOF
BatchMode=false
# String Specify the characters that will be considered for an EventBreak, by default "\n"
EventBreak="\n"
# Whether or not this input is active
Disabled=false
# Random chars that can be used to bypass previously stored states/offsets for a given file url
# Useful when the file has been rewritten and it's required for it to be processed from 0 again
Salt
# Boolean to specify if the input should be threats as a single file or a folder and all files inside should be monitored
Folder=false
# Max number of bytes to read at a time from the file
BatchSize=65536
# Since: 1.1.49
# Disable event breaking, this will present all the read bytes in the current batch.
# In combination with BatchSize it can be used to read the file in one go, this is useful to process json files
NoEventBreak=false

## Network ##
# Stanza to capture network traffic on a given local interface
[net.en0]
## Required Fields ##
# Array with the names of the rules to be applied to this input
Rules=["ff1.dstip","hash.srcip","recode.srcip(Extended,AlphaNum)"]
# Array of outputs where processed events will be sent after they go through the pipeline
# IMPORTANT: At the moment this array can only have 1 output, having 2 or more will cause random lockups
Outputs=["kv"]
# List of field processors to use when decoding data from captured packets
# Mostly in the format layer.attribute, but a few shortcuts are provided
# as they are used often (proto,src_port,src_opt, time, bytes)
# Full list available at: docs/net_fields.md
Fields=["ip.src","ip.dst","src_port","dst_port"]

# Include a _packet field with the raw captured bytes
Raw=false
## OPTIONAL FIELDS ###
```



```
# Name of the interface that we will capture packets from, should be equal to stanza name
# net.en0 --> Name = en0
Name="en0"
# Used to identify the source type for common data format type (datetime formats, timezone etc)
SourceType="net"
# Used to identify which index we prefer the data to be indexed to
Index="pt_net"
# A BPF filter to specify which packets to capture, if not defined will capture everything
# (https://biot.com/capstats/bpf.html)
# Default = ""
BpfFilter="tcp src port 80"
# Whether of not this input is active
Disabled=false

# DBConfig - struct to hold database configurations
[db.sitam]
## Required Fields ##
User="notroot"
Pass="youwishyouknew"
Host="hostofdb"
# Database driver to use, supported:
# mysql
# psql or postgres
# mssql or sqlserver
Driver="mysql"
Port=3306
Database="sitam"
# Since 1.1.51
# Defines the max number of connections to hold in pool 0 for unlimited
MaxConn=0

### SQL Database ##
# Stanza do query databases at a given periodicy
[dbin.users]
# Required Fields ##
# Name of the database stanza to use as source
Db="sitam"
# SQL Query to be run
SQL="select id,name,job from users where job = 'XPTO'"
# Array with the names of the rules to be applied to this input
Rules=["rex.csv","ff1.dstip","hash.srcip","recode.srcip(Extended,EmailAlpha)"]
# Cron expression of when to run this input, with seconds resolution.
# This avoid relying on external tools like unix cron, which has at max minute resolution
# "0 30 3-6,20-23 * * *" - in the range 3-6am, 8-11pm
# "CRON_TZ=Asia/Tokyo 0 30 04 * * *" - Runs at 04:30 Tokyo time every day"
# "@hourly" - "Every hour, starting an hour from now"
# "@every 1h30m" - "Every hour thirty, starting an hour thirty from now"
# "! 0 0 0 * * *" - "Runs at midnight everyday, and when on service start (!)"
Cron="! 0 0 0 * * *"

## Optional Fields ##
#
# Name of the stanza, usefull for logging and debugging
# Not needs for anything else
```

```
# Please name me users to match the stanza name [dbin.users], but you can actually name anything
Name="users"
# Used to identify which index we prefer the data to be indexed to
Index="pt_db_sitam_users"
# Array of outputs where processed events will be sent after they go through the pipeline
# IMPORTANT: At the moment this array can only have 1 output, having 2 or more will cause random lockups
Outputs=["kv"]
# Selected Field that will be used for ordering the processing and batching
OrderField="id"
# Boolean to disable input
Disabled=false
# Number of rows to process at a time, we default to 10000 not to overload the database with very large requests
# Also at the end of sql inputs are normally sql outputs, which when done in large thousands can have serious
impact on BD performance
BatchSize=10000
# When through prints additional debug messages
Debug=false
#disable tracking
NoTracking=false
# Field to reboot tracking
Salt=""

## WinEvent input
# Collects winevents from the local machine
[evt.setup]
# Name of the LogSource to collect
LogName="setup"

## Metrics input
# Collects system metrics from the local machine
# CPU,DISK,NET,MEMORY,Connections and Process information
[metrics.all]
Metrics=[
    "cpu", # Cpu time consumed
    "cpuinfo", # Cpu version, and spec
    "virtualmem", # Memory used/free
    "disk", # Disk IO counters
    "loadavg", # CPU load average 1,5,15m
    "net", # Net IO Counters by interface
    "connections"] # Connection,src,dst,ports,process pid, type, status

## HTTP Input
# Since: 1.1.57
# The port can be specified on System.Web.Port stanza
[http.in]
# Path where the pipeline will be pre-deployed
# If there are two equal paths the last one defined will overwrite
Path="/lessons"
# Rule pipeline to process
Rules=["rule.name"]

# Output stanzas to be used in the input stanzas
[outputs.kv]
# Format to write the output in
```

```
# "raw" - Writes the bytes was they arrived during the pipeline
# Useful for binary data transfers or pcaps
#
# "json" - Formats the ReadBuffer ([]Event) in json as an array of events
# [ {"key": "value1", "other_prop": 1}, {"key": "value_event2", "other_prop": 2} ]
#
# "kv" - Formats the ReadBuffer ([]Event) in key value format
# key="value" other_prop=1
# key="value_event_2" other_prop=2
#
# "csv" - Formats the ReadBuffer ([]Event) in csv format
# key,other_prop
# value,1
# value_event_2,2
#
# "custom" - Formats the Events using a string template defined on the Template attribute
Format = "custom"
# Template to be used on custom output formats
Template="$name is the best journalist in the whole $place"
# Magic URLs that point to the right output
# Supported Formats
# "tcp://host:port" - Open a TCP connection to <host> at <port> sends the event bytes there
# "udp://host:port" - Open a UDP connection to <host> at <port> sends the event bytes there
# "metago://host:port/path" - Uses HTTP with METAGO event encoding to <host>:<port>/<path>
# "metagos://host:port/path" - Uses HTTPS with METAGO event encoding to <host>:<port>/<path>
# "file://local_path" - Write to a local file
# "pcap://local_path" - Writes a pcap file, only to be used with network packet captures
# "stdout://" - Write to STDOUT
Urls = [ "stdout://" ]
## OPTIONAL FIELDS ##
# Only useful for file outputs, values can be "append" or "write"
Mode="append"

## Text
# Since: 1.1.59
# Static input for testing purposes
[text.name]
Text="My test to be processed"

# SourceTypes
# Since: 1.1.0
# SourceType is a string that defines the type of the source, and configures the time parser to be used
# as well as the timefield to be considered as the time of the event
[sourcetypes.sysmon]
# Required Fields
# Format of the event should be encoded when sharing with other systems
Format="METAGO"
# Field to be considered as the time of the event
TimeField="timestamp"
# Time parser to be used to parse the time field
TimeFormat="YYYY-MM-DDTHH:mm:ss"

# Stanza for internal system configuration
[system.web]
```

```
# Disables the web server API  
Disabled=true  
# Since: 1.1.57  
Port=8080
```

Annex II – Rules Spec

```
#### Common to all Rules ####
# name of the stanza, use for reference from inputs
Name
# enable debug extra verbose
Debug
# disable processing this rule
Disabled

##### FF1 #####
# Encrypts or decrypts fields using AES-FF1
# The key and tweak are randomly generated on the first run
# and saved encrypted on the local database
[ff1.cadev_sim]
## Required ##
Fields=["Cliente","ClienteNome","ClienteMorada","ClienteTelefone","ClienteEmail"]
## Optional ##
# Output fields where the encrypted inputs results from Fields will be stored.
# If nothing is provided it will default back to Fields
Dests=["Cliente","ClienteNome","ClienteMorada","ClienteTelefone","ClienteEmail"]
# Radix of the alphabet to decode
Radix=10
# "" - Default no func
# "nif" - Removes check digit before encrypt, calc check after
# "email" - Formats the output as an email
Check=""
# MaxSize allowed after encrypting, return error into ERROR_FIELD if exceded
# Default = 4096
MaxSize=4096
# "encrypt" or "decrypt"
Mode="encrypt"

##### DATE #####
# Export the current date in the defined format
[date.anonDate]
# Output format for the date
Format="2006-01-02"
# Field in the event where the date will be stored
Field="__DATE__"

##### BUCKET #####
# Takes a list of fields and performs date rounding to
# one of the following Durations
# "second", "minute", "hour", "day", "week", "month", "year"
[bucket.bin_riscos]
# List of fields to be generalized
Fields=["CondutorDataNasc","TomadorDataNascimento"]
# "second", "minute", "hour", "day", "week", "month", "year"
Duration="month"
# Format of dates / time to be processed
# follows golang clever format, 06-year, 01-month, 02-day etc
Format="2006-01-02 15:04:05"
```

```
##### ErrorsStatus #####
# Deprecated: Will be removed when Eval and CAL and mainstreamed
# Quick and brainless hack to solve a client's problem
# Is equivalent to event[Field] = event[__ERROR__] ? "OK" : "Error"
[errorstatus.anonStatus]
# Storage field
Field="__STATUS__"
# Value if no errors in the pipeline
Value="OK"
# Value if error detected on the pipeline
ErrorValue="Error"

##### CSV #####
# Parse event bytes from CSV to Event
[csv.in]
# Should we use the first line as headers?
Headers=false
# Alternative to define the column name
Fields=["column1","column2"]

##### KV #####
# Parse event bytes from Key Value format to Event
# <key> $KVSEP[$Quote]<value>[$Quote]$FieldSEP$LineSEP
# Example
# name=value age=12\n
#
# key = name
# KVSEP = "="
# Quote = ""
# FieldSEP = " "
# LineSEP = "\n"
# Raw = false
[kv.in]
# ALL fields are optional with following defaults
KVSEP="="
FieldSEP=" "
LineSEP="\n"
Quote=""
Offset=0
# If true will add field _raw with the original source line per event
Raw=false

##### Excel Parser #####
# Parse and excel file in xlsx format, each row to Event
# First row is used has headers
# Each successive row is considered value
# Input is taken from ReadBuffer.Bytes
[excel.bytes]

##### REQUIRE #####
# Deprecated: Will be replaced by a generic search / where command
# Filters events requiring an array of fields to be present, otherwise event is discarded
[require.ClienteNIF]
Fields=["ClienteNIF"]
```

```
##### HASH #####
# Apply the select Hash algorithm to a list of Fields
[hash.fields]
# Required
Fields=["name","age"]
# Optional
# Salt value to apply, always the same, doesn't vary by run, field or event
# Though that could be extend to vary by event without much pain
Salt=[]
# algorithm to apply, one of "MD5","SHA1","SHA224","SHA256","SHA384","SHA512"
Algo string
# Radix of the alphabet to use for decoding the field string to bytes
Radix int

##### Encrypt #####
# Apply the select Cipher algorithm to a list of Fields
# The key and tweak are randomly generated on the first run
# and saved encrypted on the local database
[encrypt.fields]
# Required
# Array with list of fields to be processed
Fields=["name","age"]
# Optional
# Salt value to apply, always the same, doesn't vary by run, field or event
# Though that could be extend to vary by event without much pain
Salt=[]
# algorithm to apply, one of "AES-128-CBC"
Algo string
# Radix of the alphabet to use for decoding the field string to bytes
Radix int
# Output fields where the encrypted inputs results from Fields will be stored.
# If nothing is provided it will default back to Fields
Dests=["ClientName","ClientAge"]
# "encrypt" or "decrypt"
Mode string

##### REX Regular Expression #####
# Applies a list of regular expressions to a given field,
# only the first to match will extract any new fields.
# Fields can be extracted using named captures or traditional regex grouping
# (?P<name>)[^,]+) -> would extract to the current event {"name": "<captured_text>"}
# ([^,]+) -> would extract to the current event {"0": "<captured_text>"}
[rex.myregexes]
# Required
# List of patterns to process by order on each event going through the pipeline
Patterns=["(?P<name>)[^,]+","can't stop (?P<program>[^ ]+) from crashing and (?P<sideeffect>[^ ]+)"]
# Optional
# Field on which to apply the regular expressio, if nothing is given it will be checked against
# the internal Bytes on the ReadBuffer
Field="_raw"
# Optional
# Since: 1.1.48
# Allows for the grep -v logic where the regex is inverted
```

```
Invert=false
# Optional
# Since: 1.1.48
# Specify the run mode, filter the Events or Extract Fields, by default extracts fields
Filter=false

#### Excel Out ####
# Converts the current list of events into an Excel xlsx file format
# and saves it into the current rb.Bytes
[excelout.savefile]
# Optional
# Sheet name on the saved Excel file by default Sheet1
SheetName="Sheet1"

#### Base64in ####
# Decodes the rb.Bytes in the current buffer from Base64 with StandardEncoding and saves it back to rb.Bytes
[base64in.name]

#### Base64out ####
# Encodes the rb.Bytes in the current buffer from Base64 with StandardEncoding and saves it back to rb.Bytes
[base64out.name]

#### ipmask ####
# Apply a netmask to an IP address as means to anonymize the sensitive lower octets
# ipmask("192.168.0.1","255.255.0.0") -> "192.168.0.0"
[ipmask.srcip]
# List of fields to perform ipmasking on
Fields=["srcip"]
# Subnet to apply the ipmasking
Mask="255.255.0.0"

#### DB Columns ####
# DON'T USE - Work in progress
#
# Will extract the fields and their types from selected database and table
# still working out how this will be used by PIIWeb
#
# DON'T USE - Work in progress
[dbcOLUMNS.database]
DB="database stanza name"
Field="tablename"

#### Recode ####
# Change the encoding of a string from a source encoding to a target encoding
# Very basic alphabet recoding doesn't change the string native utf-8 encoding
# This allows to convert between bases hex to dec etc, up to base 170.
# ExtendedAlpha is 170 chars long and the biggest encoding scheme defined.
# Optionally users can define their own input and output alphabet
[recode.fields]
## Required Fields ##
# Array with list of fields to be processed
Fields=["name","age"]
# Internal Alphabet to decode the fields
# Name or list of runes to be considered for alphabet
```


Page 78

```
Min=0
# The maximum int to generate
Max=1000

#### Str ###
# Use a string template to generate a new string
[str.new_string]
# Field where to store the value
Dest="newstring"
# Template of the string to generate
# Variables from the current event can be accessed through $var or ${var}
# ${var} is required for attributes with non valid identifier chars
# C is a valid identifier char if:
# (c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c == '_') || (c == '-')
# summing it up alphanum + _ + -
Template="text and $variable, even ${weird value names}"

#### ESOOut ###
# Saves the current pipeline result to an elastic search database
[esout.savetoelastic]
## Required ##
# List of fields to save to elastic
# this will save {"name": <namevalue>, "age": <agevalue>}
Fields=["name","age"]
# ES index
Index="main"
Host="EShost"
Port=9200
## Optional ## authentication
# User and Pass
User=""
Pass=""

#### Lookup ###
# Static lookup of a field in a map of values
# In this example it will lookup the value of Field user_id,
# if it matches a key in Map it will save the value in the Map into Event[user_name]
[lookup.values]
# Field to check value
Field="user_id"
# Field to save the lookup
Dest="user_name"
# Static map to check the Field value
Map={
  1= "luis",
  2= "josh",
  3= "maria"
}

#### CSVLookup ###
# Perform a join between existing column and a CSV file
#
# Example
# a.csv
```

```
# user_id,user_name
# 1,jose
# 2,manuel
#
# Event
# {"user_id":1} | csvlookup(user_id,[user_name]) -> {"user_id":1, "user_name":"jose"}
[csvlookup.values]
# Field to check value
Field="user_id"
# Path of CSV file
Path="a.csv"
# Fields to export back to the Event, if Outputs is not defined it will fallback to all fields in the CSV
Outputs=["user_name"]

#### IF ###
# Conditional processing of flows
# If the evaluation of Condition returns true it will execute next the rule in the True branch,
# else the rule whose name is on the False branch, if one is provided
[if.filter]
# Condition to be evaluated
# Check docs/cal.md for details on the supported scripting language
Condition="$name != nil && $name != \"\""
# Name of the rule to be executed if the condition is true
True="ff1.name"
# Name of the rule to be executed if the condition is false
False=""

#### EVAL ###
# Evaluate a scripting language CAL and store the result in user configured field specified on the Var attribute
# Example from below
# 1) Define fib function
# 2) call that function with the value of $age
# 3) save the result in Event["out"]
[eval.cal]
# Cal expression to evaluate
Expr=""
def fib(n){
  if (n<2) {
    n
  }else {
    fib(x-1) + fib(x-2)
  }
}
fib(int($age))
""
# Variable to save the result
Var="out"

#### DBOUT Save to DB ###
# Execute an Insert or Update statment constructed from a SQL template with optional Params that will be
# retrieve from each Event
[dbout.cadev_out_sim]
# Database definition where to connect to
DB="sitam"
```

```
# Truncate the output if db column can't store it all, instead of letting the DB deal with it
Truncate=false
# Prints the SQL instead of executing updates implies KeepStatus = false
DryRun=false
# Prints the prepared statement as SQL query instead of placeholders with args
DrySQL=false
# Params to be used as query parameters
Params=["Cliente","ClienteNome","ClienteNIF","ClienteDataNascimento","ClienteMorada","ClienteTelefone","C
lienteEmail","__STATUS__","__DATE__","Simulacaold"]
# SQL to be executed
Query="UPDATE [dbo].[Simulacoes] SET [Cliente] = ? , [ClienteNome] = ?, [ClienteNIF] = ?,
[ClienteDataNascimento] = ?, [ClienteMorada] = ?, [ClienteTelefone] = ?, [ClienteEmail] = ?,
[estadoAnonimizacao] = ?, [dataAnonimizacao] = ? WHERE [Simulacaold] = ?"
# Whether or not to keep tracking of the processed rows
KeepStatus=true

#### DBLOOKUP ###
# Lookup a set of columns from a Database based on an array of common values
#
# Example
# If we have Event{user_id: 1} and a database with
# table users
# user_id,name
# 1,Luis
# Invoking this function with ["user_id"] as Params would yield:
# dblookup(user_id) -> Event{user_id:1, name: "Luis"}
[dblookup.sim_by_id]
# Database definition where to connect to
DB="cadev"
# Params to be used as query parameters, and source values from Event to perform the lookup
Params=["Simulacaold"]
# SQL to be executed
Query=""
SELECT
Simulacaold,Cliente,ClienteNome,ClienteNIF,ClienteDataNascimento,ClienteMorada,ClienteTelefone,ClienteE
mail
FROM [dbo].[Simulacoes]
WHERE (estadoAnonimizacao='OK') AND Simulacaold IN (?) ""

#### STRING #####
# Since 1.1.40
# Action to perform common string operations
# Operations | ArgCount | Args | Return
# TRUNCATE | 1 | int - max number of runes | string - s[:max]
# CONTAINS | 1 | string - substring to search | bool - true if field contains substring
# HASPREFIX | 1 | string - prefix string | bool - true if field starts with substring
# HASSUFFIX | 1 | string - suffix string | bool - true if field ends with substring
# INDEX | 1 | string - substring to search | int - index of first substring occurrence
# RINDEX | 1 | string - substring to search | int - index of last substring occurrence
# LOWER | 0 | | string - field in lowercase
# UPPER | 0 | | string - field in uppercase
# TRIM | 1 | string - substring to trim | string - field without substring in prefix or suffix
# TRIMLEFT | 1 | string - substring to trim | string - field without substring in prefix
# TRIMRIGHT | 1 | string - substring to trim | string - field without substring in suffix
```

```
# SLICE      | 1 or 2 | start, end int      | string - s[start:end]

[string.username]
Op="Truncate"
Fields=["username"]
Args=[23]

#### Replace
# Since 1.1.61
# Replace a regex match or list of matches with a static string
[replace.str]
# In revision 1.63.0 this field was renamed from Field to Fields and contains the array of fields to apply the
replace to
Fields=["query"]
# Regex to select the relevant parts
Regex='(d+)'
# Replacement Value
Value="."
# Destination Fields, optional, if omitted will fallback to Fields
Dests=["query"]

### Pipelines ###
# Since 1.1.40
# Allow for inline definition of pipelines
# currently useful for IF rules
[if.havefun]
True="pipeline.name(csv.rule.name,require.rule.name2)"

### JSONARY ###
# Unmarshal json string with the format array of objects [{..},{...}] into rb.Events []map[string]interface{}
# Since 1.1.48
[jsonary.in]
# Field where to read the string json from
Field="name"

### JSONROW ###
# Unmarshal json string with the format array of objects {..}\n{...}\n... into rb.Events []map[string]interface{}
# Since 1.1.48
[jsonrow.in]
# Field where to read the string json from
Field="_raw"
# Since 1.1.56
# Specify if the source field should be keep on the Event or removed
KeepSource=false

### JSET ###
# Since: 1.1.49
# Set values in arbitrarily deep json
# Below is a quick overview of the path syntax, for more complete information please check out GJSON Syntax.
# A path is a series of keys separated by a dot. A key may contain special wildcard characters '*' and '?'.
# To access an array value use the index as the key. To get the number of elements in an array or to access a
child path, use the '#' character.
# The dot and wildcard characters can be escaped with '\'.
# {
```

```
# "name": {"first": "Tom", "last": "Anderson"},
# "age": 37,
# "children": ["Sara", "Alex", "Jack"],
# "fav.movie": "Deer Hunter",
# "friends": [
#   {"first": "Dale", "last": "Murphy", "age": 44, "nets": ["ig", "fb", "tw"]},
#   {"first": "Roger", "last": "Craig", "age": 68, "nets": ["fb", "tw"]},
#   {"first": "Jane", "last": "Murphy", "age": 47, "nets": ["ig", "tw"]}
# ]
# }

# "name.last"      >> "Anderson"
# "age"           >> 37
# "children"      >> ["Sara", "Alex", "Jack"]
# "children.#"    >> 3
# "children.1"   >> "Alex"
# "child*.2"     >> "Jack"
# "c?ildren.0"   >> "Sara"
# "fav\movie"    >> "Deer Hunter"
# "friends.#.first" >> ["Dale", "Roger", "Jane"]
# "friends.1.last" >> "Craig"

# You can also query an array for the first match by using #(...), or find all matches with #(...)#.
# Queries support the ==, !=, <, <=, >, >= comparison operators and the simple pattern matching % (like) and
# !% (not like) operators.
# friends.#(last=="Murphy").first >> "Dale"
# friends.#(last=="Murphy").first >> ["Dale", "Jane"]
# friends.#(age>45)#.last >> ["Craig", "Murphy"]
# friends.#(first%"D*").last >> "Murphy"
# friends.#(first!%"D*").last >> "Craig"
# friends.#(nets.#(!="fb"))#.first >> ["Dale", "Roger"]
[jset.value]
# Field source to retrieve json from, can be the special value _raw to retrieve from rb.Bytes or anything else to
# retrieve from each event
Field="JsonSourceField"
# Field to retrieve the value from
# if Field=RawField then we will retrieve the value from rb.Meta else from rb.Events[i]
Value="FieldToRetriveValue"
# Expression to set
Path="name.first"

# JGET
# Since: 1.1.49
# Retrieve a value from a JSON field that is arbitrarily deep
[jget.value]
# Field source to retrieve json from, can be the special value _raw to retrieve from rb.Bytes or anything else to
# retrieve from each event
Field="JsonSourceField"
# Field where to store the result
# if Field=RawField then we will retrieve set value on rb.Meta[$Dest] else on rb.Events[i][$Dest]
Dest="FieldToRetriveValue"
# Expression to set
Path="name.first"
```

```
# JDEL
# Since: 1.1.49
# Delete a value from a JSON field that is arbitrarily deep
[jdel.value]
# Field source to retrieve json from, can be the special value _raw to retrieve from rb.Bytes or anything else to
retrieve from each event
Field="JsonSourceField"
# Expression to delete
Path="name.first"

# Flatten
# Since: 1.1.56
# Take a deep object like struct and flatten it out
# Example:
# Maps -> {a: {b: 1, c: 2}} -> {a.b: 1, a.c: 2}
# Arrays -> [{a: 1}, {a: 1, b: 2}] -> {"0.a": 1, "1.a": 1, "1.b": 2}
[flatten.field]
# Field name to flatten, if the special __all__ field is specified then all record fields are flattened out
Field="__all__"

# MVMAP
# Since: 1.1.50
# Must appear after Rule has been defined
# Apply a rule to all elements of an array and return a new array with same cardinality
[mvmap.name]
# Field where to apply the map function
Field="ips"
# Rule to call for each element inside the array / slice
Rule="clean.ips"

# JSONP
# Since: 1.1.56
# Parse a json str to an Event struct
[jsonp.raw]
# Field to use as the source of json string
Field="_raw"
# If true the select source field will remain in the event,
# If false it will be removed
KeepSource=false

# FileIn
# Since: 1.1.56
# Read a file at a given path and copy contents to Rb.Bytes
# Open mode is read
# All bytes are fully read in one go
[filein.raw]
# The filepath to monitor, can be absolute or relative to work dir
Path="in.csv"
# File encoding, by default utf-8 if nothing else selected
# Full list available at docs/encode_formats.md
Encoding="utf-8"
# String Specify the characters that will be considered for an EventBreak, by default "\n"
```

```
EventBreak="\n"

# FileOut
# Since: 1.1.56
# Save Rb.Bytes to a file at a given path
[fileout.out]
# The filepath where to save, can be absolute or relative to work dir
Path="out.csv"
# Format to write the output in
# "raw" - Writes the bytes was they arrived during the pipeline
# Useful for binary data transfers or pcaps
#
# "json" - Formats the ReadBuffer ([]Event) in json as an array of events
# [ {"key": "value1", "other_prop": 1}, {"key": "value_event2", "other_prop": 2} ]
#
# "kv" - Formats the ReadBuffer ([]Event) in key value format
# key="value" other_prop=1
# key="value_event_2" other_prop=2
#
# "csv" - Formats the ReadBuffer ([]Event) in csv format
# key,other_prop
# value,1
# value_event_2,2
#
# "custom" - Formats the Events using a string template defined on the Template attribute
Format = "custom"
# Template to be used on custom output formats
Template="$name is the best journalist in the whole $place"
# Magic URLs that point to the right output
## OPTIONAL FIELDS ##
# Only useful for file outputs, values can be "append" or "write"
Mode="append"

#### mask ####
# Since: 1.1.58
# Apply a mask to an string as means to anonymize the sensitive chars
# mask("My Name Is: Charlote","#####",true) -> "My Name Is: "
# mask("My Name Is: Charlote","#####",false) -> "My Name Is: _____"
[mask.name]
# List of fields to perform masking on
Fields=["name"]
# Mask to apply
Mask="#####"
# If true the final value will have the size of the mask value
# If false the final value will have the size of the source field value
TrimRight=false
```