HORIZON 2020
Information and Communication Technologies
Integrating experiments and facilities in FIRE+

Deliverable D3.3
Privacy Tools 1st Iteration

Grant Agreement number:	**687884**
Project acronym:	**F-Interop**
Project title:	**FIRE+ online interoperability and performance test tools to support emerging technologies from research to standardization and market launch** **The standards and innovations accelerating tool**
Type of action:	**Research and Innovation Action (RIA)**
Project website address:	**www.finterop.eu**
Due date of deliverable:	**31/10/2017**
Dissemination level:	**PU/CO**

This deliverable has been written in the context of the Horizon 2020 European research project F-Interop, which is supported by the European Commission and the Swiss State Secretariat for Education, Research and Innovation. The opinions expressed and arguments employed do not engage the supporting parties.


Co-funded by the European Union


Co-funded by the Swiss Confederation

Document properties

Responsible partner	University of Luxembourg
Author(s)/editor(s)	Luca Lamorte (UL), Ion Turcanu (UL), Eunah Kim (DG)
Version	1.0
Keywords	Test tool, Privacy

Abstract

The deliverable D3.3 aims to describe the first release of the Privacy Test Tool, one of the testing tools available in F-Interop Platform, which has been developed within WP3, and specifically as an output of the Task 3.2.

Starting from the general definition of a Test Session, defined in Deliverable D1.1, and adopted as a lifecycle guideline for any F-Interop testing tool, this deliverable describes how each step has been adopted by the Privacy Test Tool followed by the lifecycle guidelines of F-Interop testing tools.

The deliverable also contains the design principles of the Privacy Test Tool, with the related description of each module that is part of it.

Moreover, configurations, flow messages and all the integration mechanisms used to deploy the tool in the F-Interop architecture are also part of this manuscript.

This first version of the Privacy Test Tool provides means to the F-Interop user to perform privacy analysis on the IoT traffic generated while connected with other remote components. The privacy assessment report is provided in a visual form illustrating issues and threats detected that helps the engagement and the trustiness of the platform from the user perspective.

Table of Contents

List of Figures

List of Tables

List of Acronyms

ABC	Attribute Based Credential
API	Application Program Interface
CA	Consortium Agreement
CoAP	Constrained Application Protocol
ComSoc	Communications Society
DESCA	Development of a Simplified Consortium Agreement
DHCP	Dynamic Host Configuration Protocol
DHT	Distributed Hash Tables
DNS	Domain Name System
DNSSec	Domain Name System Security Extensions
DPA	Data Protection Authorities
DPO	Data Protection Officer
EC	European Commission
ENISA	European Union Agency for Network and Information Security
ETSI	European Telecommunications Standards Institute
EU	European Union
FP7	Seventh Framework Programme
GA	Grand Agreement
GA	General Assembly
GPS	Global Positioning System
HTTPS	Hypertext Transfer Protocol Secure
ICT	Information and Communication Technologies
ID	Identifier
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IERC	European Research Cluster on the Internet of Things
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
IPC	Intellectual Property Committee
IPM	IPR Monitoring and Exploitation Manager
IPR	Intellectual Property Rights
IPSEC	Internet Protocol Security
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISO	International Standards Organization
ISP	Internet Service Provider
IT	Information Technology
ITU	International Telecommunication Union
KPI	Key Performance Indicator
LSPI	Legal, Security and Privacy Issues
MAC	Media Access Control
MI	Mandat International
MSc	Master of Science

M2M	Machine to Machine
NFV	Network Functions Virtualization
OASIS	Organization for the Advancement of Structured Information Standards
OECD	Organization for Economic Cooperation and Development
OS	Operating System
OSN	Online Social Network
PC	Project Coordinator
PCP	Partner Contact Person
PDPO	Personal Data Protection Officer
PERT	Program Evaluation Review Technique
PhD	Doctor of Philosophy
PM	Person Month
PMB	Project Management Board
PPR	Periodic Progress Report
PRAAT	Privacy Risk Area Assessment Tool
P&T	Post & Telecom
QoS	Quality of Service
RAND	Reasonable and Non Discriminatory
RFC	Request For Comments
R&D	Research & Development
SME	Small Medium Enterprise
SMS	Short Message Service
SOTA (or SoA)	State Of the Art
SSL	Secure Sockets Layer
SDN	Software Defined Networking
TC	Technical Coordinator
TCP	Transmission Control Protocol
TL	Task Leader
TLS	Transport Layer Security
Tor	The Onion Router
TRL	Technology Readiness Level
UK	United Kingdoms
UN	United Nations
UNCTAD	United Nations Conference on Trade and Development
UPRAAT	Universal Privacy Risk Area Assessment Tool
URL	Uniform Resource Locator
US	United States
VoIP	Voice over Internet Protocol
WES	Women's Engineering Society
WiTEC	Women in science, Engineering and Technology
WoT	Web of Trust
WP	Work Package
WPL	Work Package Leader
W3C	World Wide Web Consortium
XML	Extensible Markup Language

1 Introduction

1.1 About F-Interop

F-Interop is a Horizon 2020 European Research project, which proposes to extend the European research infrastructure (FIRE+) with online and remote interoperability and performance test tools supporting emerging technologies from research to standardization and to market launch. The outcome will be a set of tools enabling:

- Standardization communities to save time and resources, to be more inclusive with partners who cannot afford travelling, and to accelerate standardization processes;
- SMEs and companies to develop standards-based interoperable products with a shorter time-to-market and significantly lowered engineering and financial overhead.

F-Interop intends to position FIRE+ as an accelerator for new standards and innovations.

1.2 Deliverable Objectives

1.2.1 Work package Objectives

- Research and develop performance test tools.
- Research and develop tools for privacy risk assessment
- Research and develop spatial representing tools to support experimenters.
- Research and integrate testing tools with network virtualization technologies such as OpenFlow / OpenDayLight based SDN / NFV environments.

1.2.2 Task Objectives

Work: Task 3.2 will design methods for privacy analysis of the data exchanged while running different kind of tests on the F-Interop platform. First, the State-of-Art of traffic analysis on encrypted data flows will be studied, in order to check potential suitable solutions for F-Interop. Then, tools for checking how privacy issues may raise due to information leakage will be developed. In detail, it will be investigated how an adversary may get “sensitive” information (e.g., results of a test running on the shared platform) by passively observing patterns of the IoT communication.

Roles: UL will lead the task with the support of MI.

Outcome: A tool for assessing the privacy leakage from IoT communication.

1.2.3 Deliverable Objectives and Methodology

1.2.3.1 Deliverable Objectives

The deliverable D3.3.1 – Privacy Test Tool 1st iteration is the description of the first release of the Privacy Test Tool as output of the Task 3.2. This report contains the first version of the F-Interop Privacy Test Tool and the required key enablers. A first version of the privacy assessment report concerning personal data is presented.

1.2.3.2 Deliverable Methodology

One of the key features of the F-Interop Platform is to enable Remote and Online communications among IoT components. In some scenarios, called Location Models in F-Interop, the public Internet will be used to exchange data. Therefore, it is essential to avoid sharing information that may be confidential or sensitive for the owner (e.g., individual, company, organization) of the connected resources. There is a need to protect any traffic exchanged and to provide an acceptable level of security and privacy.

On the other hand, users may have different views of privacy and security and they might want to carefully check what is shared during a test session. Hence, the platform should provide means to check if a certain privacy and security level is satisfied.

Following the guidelines defined in Deliverable D1.2 (Privacy by Design Report), a State-of-the-Art study regarding IoT privacy issues has been conducted. Based on that, we selected one of the most popular emerging IoT protocols, named CoAP, to investigate vulnerabilities during a data exchange session. In this first iteration, the Privacy Test Tool has been developed implementing algorithms to "flexible" detect privacy issues.

The Privacy Test Tool follows the generic F-INTEROP Test Tool Design, a common infrastructure used to encapsulate the tool business logic within the F-Interop platform. Such design is based on the concept of "F-Interop session". This is a set of actions that a user (called the F-Interop-User) has to perform in order to execute Remote or Online tests in the F-Interop-Platform.

These steps are summarized in the Table 1 of the deliverable D1.1 which is attached below.

Table 1 - F-Interop Session

Step	Action	Description
0	FI-User authentication and authorization. IUT registration/identification	FI-User authenticates in a secure way (prior FI-User registration needed) in FI-Platform. FI-User needs to be authorized to use FI-Platform resources. FI-User identifies which IUT he/she will test (prior IUT registration needed).
1	Test suites discovery and selection	FI-User starts by discovering the available test suites and by selecting the one he/she wants to execute
2	Resource description	FI-User specifies/selects resources in the F-Interop Platform that are needed for his/her F-Interop session including the location models 1, testing tools, libraries, etc. During this phase FI-Platform may request information from FI-User or provide information to FI-User for a coherent selection of the required resources
3	Resource reservation	The resources selected in the previous step are actually reserved.
4	Resource provisioning, configuration and session setup	The instantiation of the F-Interop-Platform resources that fit best with the FI-User needs is done.
5	Test execution	The online F-Interop test campaign is launched and the selected (executable) test suites are executed against the IUTs.
6	Results analysis and report	Test execution information is analysed. The test results and verdicts are provided together with explanations in case of FAIL or INCONCLUSIVE verdicts or something wrong happened. A report can be provided under request if, for example, the FI-User wants to apply for a certification/labelling program.
7	Session storage	Storage of the F-Interop session information (Session-id, User-id, FI-User's IUT-id, IUTs' version, test description, test version, testing tool, test log and results, etc.). This has to remain accessible beyond the F-Interop session for the involved parties

This document focus specifically on the **Step 5 (Test Execution)** and **Step 6 (Result analysis and report)** of an F-Interop Session.

The testing tool architecture has been discussed in details during the WP/integration meetings allowing agreement on several points including the initial implementation design and the interfaces/APIs/standards to be used among the various components.

Such agreement has been reported in D3.1, that contains the high-level description of the tools developed in WP3. Among those, there is also the privacy tool, that shares this general architecture integration as described in Figure 1.

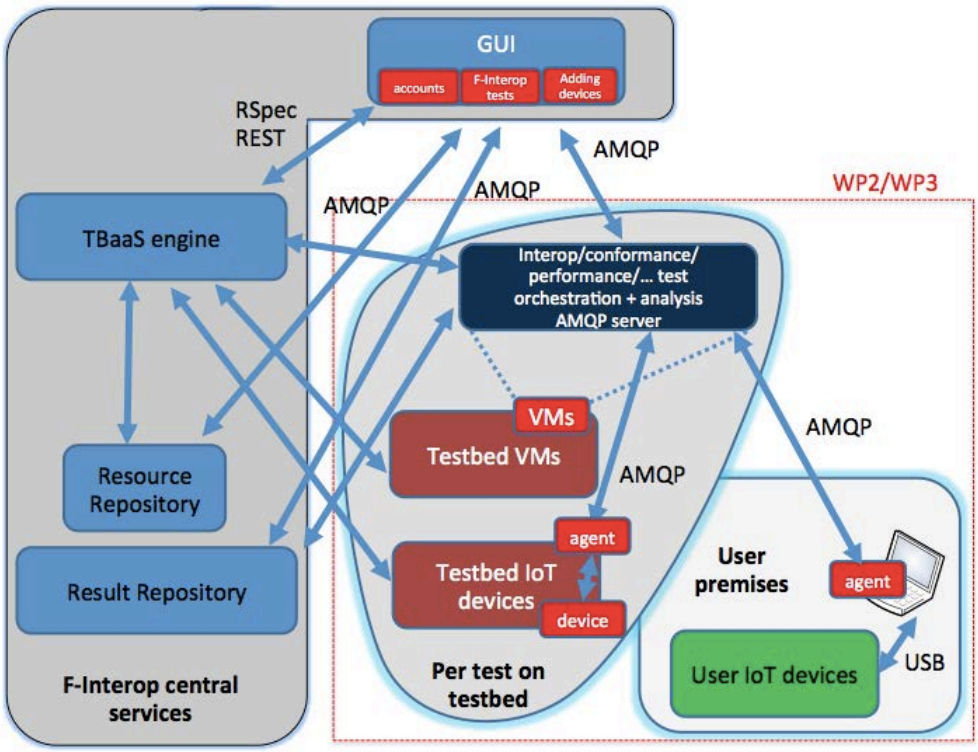

Figure 1 - High Level WP3 Building Blocks

The following chapters aims to describe in details the privacy testing tool architecture, its components and other auxiliary configurations used to perform the privacy assessment.

2 Privacy Test Tools Design

In the current implementation of the Privacy Test Tool, we consider only one of the most used application protocols for constrained devices: CoAP. The framework however is designed to make possible easy integration with other protocols. Also, the modular approach allows easy future extensions of the framework itself. The following sections describe the generic design of the framework and the functional modules of the Privacy Test Tool (further called: "subcomponents") without going into specific details of the functions provided by each module. This information is presented in detail in the Chapter 3 "Privacy Test Modules Functionality".

2.1 Privacy Test Tool Design

2.1.1 General Principles

The primary goal of the privacy test tool is to define automatic methods to detect privacy and confidential data leaks while different kind of tests are executed on the F-Interop platform. Differently from other testing tools which are devoted to test Performance, Interoperability and Compatibility of some IoT protocols, this tool wants to identify the compliance with the current European Regulation in terms of data management, increasing the trustiness of the platform while communicating with the public internet.

For this reason, a general framework has been designed to **match patterns** in the data payload of IoT protocols in order to detect what is considered **personal** and/or **private**. Such categorization may change according to the context and the user needs. Therefore, configurations and adaptations should be taken into account while applying some matching criteria.

The framework adopts an incremental approach by using plugins. These are components in charge of defining the privacy “patterns” and the best strategy to detect them.

Following this approach, the Privacy Test Tool can easily integrate new application protocols, and, simultaneously, tune or add other patterns over the time.

Such flexibility gives to the Privacy Test Tool the possibility to fit any needs for an F-Interop User, considering the specific requirements of the test he/she wants to perform and adapting the detection of what would be for him/her the meaning of “not to be disclosure” data.

Customization and configuration of the framework at run time will be integrated to better tune each testing session.

2.1.2 Generic Design

The Privacy Test Tool is a framework composed by multiple components, each of them being responsible for a different aspect of the global privacy assessment. In the current implementation (the first release), the tool is composed by the following building blocks:

- F-Interop Agent
- Packet Sniffer
- Packet Dissector
- Test Coordinator
- Privacy Tool

Some of them are working as standalone processes, while others are just logical building blocks in charge of performing specific tasks during the execution of the privacy test. Altogether, they are producing the final output of the tool – a privacy assessment report.

Figure 2 shows which are the software components involved in a generic F-Interop test session. These platform blocks interact with each other using Restful APIs or AMQP messages.

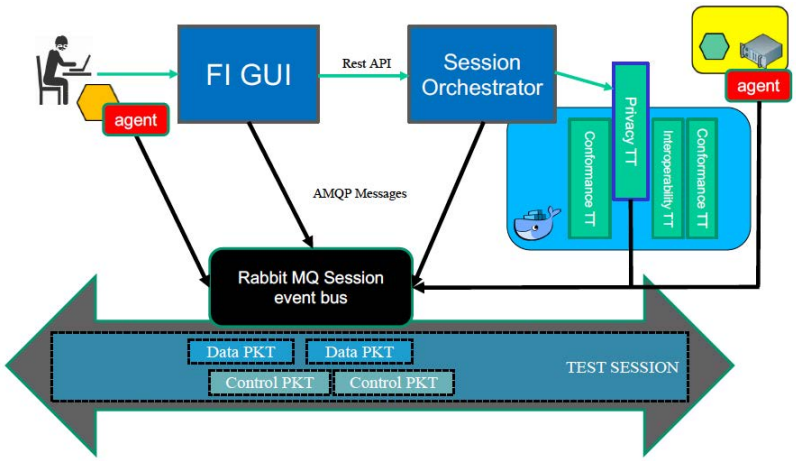

Figure 2 - F-Interop Privacy Testing tools session execution modules

Following the described flow, an F-Interop User can, first of all, select the Test Tool to be used, then a (remote) test session is created. From that moment on, control and data packets are exchanged among different resources involved in the test.

Such interactions are described in the following paragraphs of this chapter.

2.1.3 The Communication Bus

The modularity of the F-Interop architecture requires a flexible approach regarding the data exchange among different components. This is solved through an “Event Bus”, a mechanism that provides a secure channel where all the communication is done, including control messages, raw data packets and logs. As a technological solution, F-Interop has adopted RabbitMQ, which provides all capabilities required by the platform.

In particular, RabbitMQ acts as a secure message broker among all components connected through encrypted channels. It implements the **Advanced Message Queuing Protocol** (AMQP), an open standard application layer suitable for such kind of distributed architecture.

Each AMQP message exchanged contains a **routing key** and a **topic** which indicates how to route that message to the relevant input queues of the component (see Figure 3 - Example of routing messages).

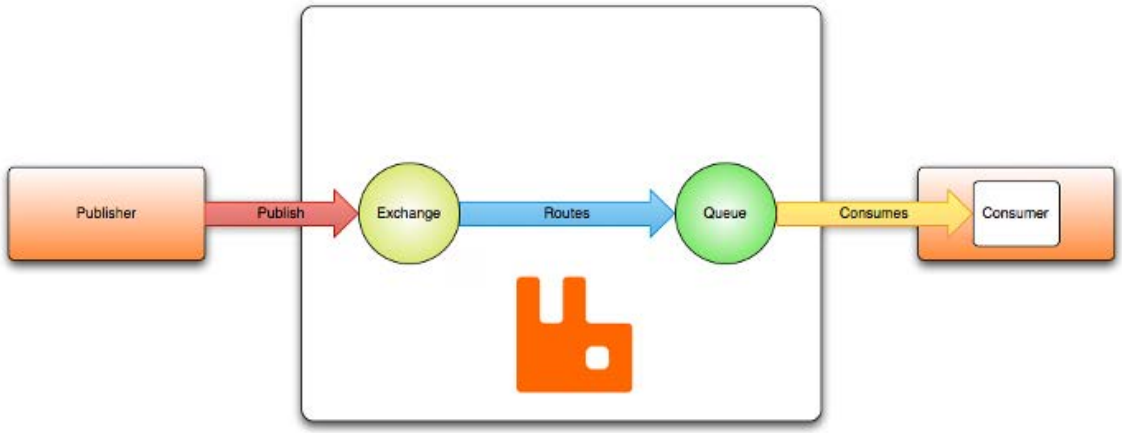

Figure 3 - Example of routing messages

In F-Interop, messages are of two types: **control plane** and **data plane**.

The **control plane** messages relate to the management of an ongoing test session: e.g. start a sniffer, signal the start/end of a test case, etc.

The **data plane** messages contain the raw data exchanged between the IUTs.

This architecture is modular and scalable by design. Components can be added/deleted from the event bus without requiring further coordination. Different components can be run on different (virtual) machines to ensure scalability. Moreover, components can be written in different programming languages, giving the freedom to adapt any tool to this architecture.

For instance, the Privacy Test Tool modules use the Event Bus to exchange messages to talk with the other components of the platform. Each of the privacy subcomponents is a separate process that maintains its own connection to the AMQP session for the whole test session.

When the tool is instantiated by the F-Interop framework, specifically by the Session Orchestrator (SO), as showed in Figure 2, the parameters necessary to connect to the Event Bus are passed through environment variables. These are:

- **Host** and **Port** of the AMQP message broker
- AMQP **vhost** used by the session
- AMQP credentials (**username** and **password**)
- AMQP **exchange** name (default is amqp.topic)

F-Interop framework uses AMQP vhosts as the mechanism to isolate multiple test sessions that can exist within the framework. All control plane communication among modules is performed within specified vhost only. Since access to each vhost is authorized with individual credentials, applications running in the other test sessions have no access to it, ensuring the confidentiality of the test execution.

Figure 4 - F-Interop remote testing architecture shows the complexities of doing an online and remote test session in F-interop. The event bust, the central block, enables the communication and the orchestration among the different modules involved.

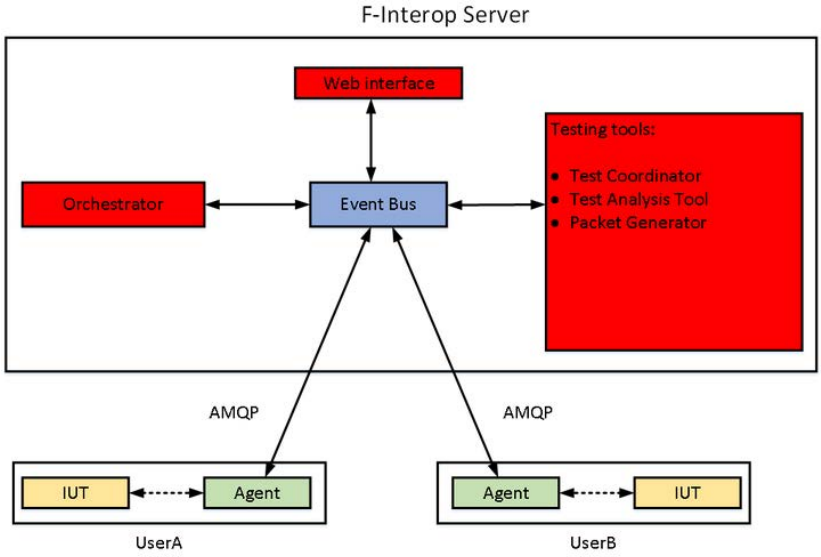

Figure 4 - F-Interop remote testing architecture

2.2 Instantiation Process

The instantiation process of the Privacy test tool is mostly identical to the instantiation process of other tools provided by the F-Interop platform such as interoperability, conformance and performance.

Through the Session Orchestrator (SO) API [1], usually triggered by the Graphical User Interface (GUI), an instance of the test tool is launched with all required configurations (see 2.1.3). These are necessary to access to the test session created by the SO. The details about the creation of the instance are reported in 2.2.2. To generalise the spawning process, each testing tools MUST be provided as a Docker container [2].

2.2.1 Privacy Test Tool Container

The Privacy Test Tool is provided in the form of a **Docker Container** [2], which encapsulates all the software required to perform the tool's task, as well as the configurations necessary to authenticate and exchange messages through the session created in the Event Bus for that particular instance. It's up to the developer of the tool to maintain and upgrade it according to the Platform changes.

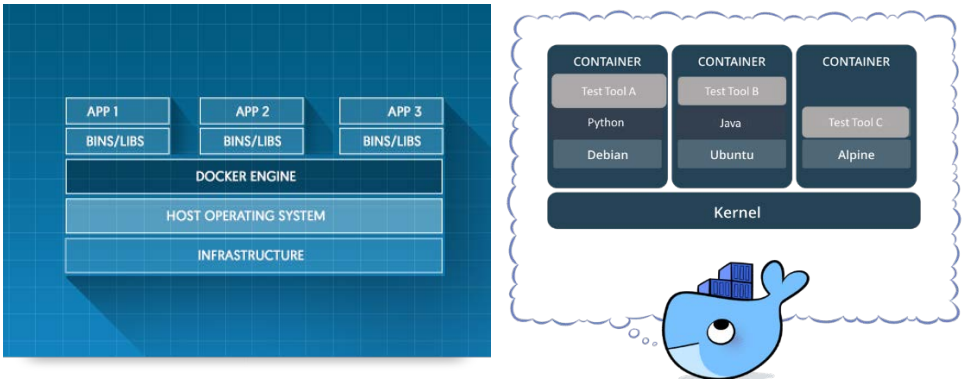

Figure 5 - Docker Container

The idea behind the Docker Container, as showed in Figure 5, is that each testing tool can be created once, but instantiated as many times as required by any F-Interop User.

In order to be executed by the F-Interop framework, the container must be **built** and **registered** in the F-Interop's Docker repository under a name in specific format, in our case:

- **testing_tool-privacy** (as version-less name)
- **testing_tool-privacy-v<VERSION>** (with version specified)

The source code for the privacy test tool [3] provides a **Dockerfile**, the configuration file used by Docker to build the container image. The Dockerfile used to build the Privacy Test Tool container is described in Appendix 6.1.1

In addition, the Privacy Test Tool provides two configuration scripts for the Session Orchestrator itself:

- **index.json**: a file containing all the necessary metadata to describe the tool, including a tool-specific configuration UI template.
- **supervisor.conf.j2** : a template for the Supervisor configuration, describing how the tool can be instantiated/launched

These two files are located separately in a folder[1], usually named according to the tool's task (for instance **privacy** for the privacy test tool) that is located in this path of the Session Orchestrator deployment:

<SO_folder>/templates/f-interop/

See the Annex 6.1.2 for the detailed description of the Orchestrator configuration scripts.

[1] **Note:** in the future releases of the F-Interop platform, it is planned to access these files via the tool's container (as attached metadata), instead of accessing static files.

2.2.2 Orchestration Process with Privacy Test Tool

In the current implementation of the F-Interop framework, the orchestration of the test session is a complex communication process between the User interface (UI), Session Orchestrator (SO) and the Test Tool (TT). The goal of the orchestration is to launch the selected testing tool, provide it with configuration and parameters to access the common communication bus (AMQP session).

In detail, the orchestration process involves the following steps as described in Figure 6:

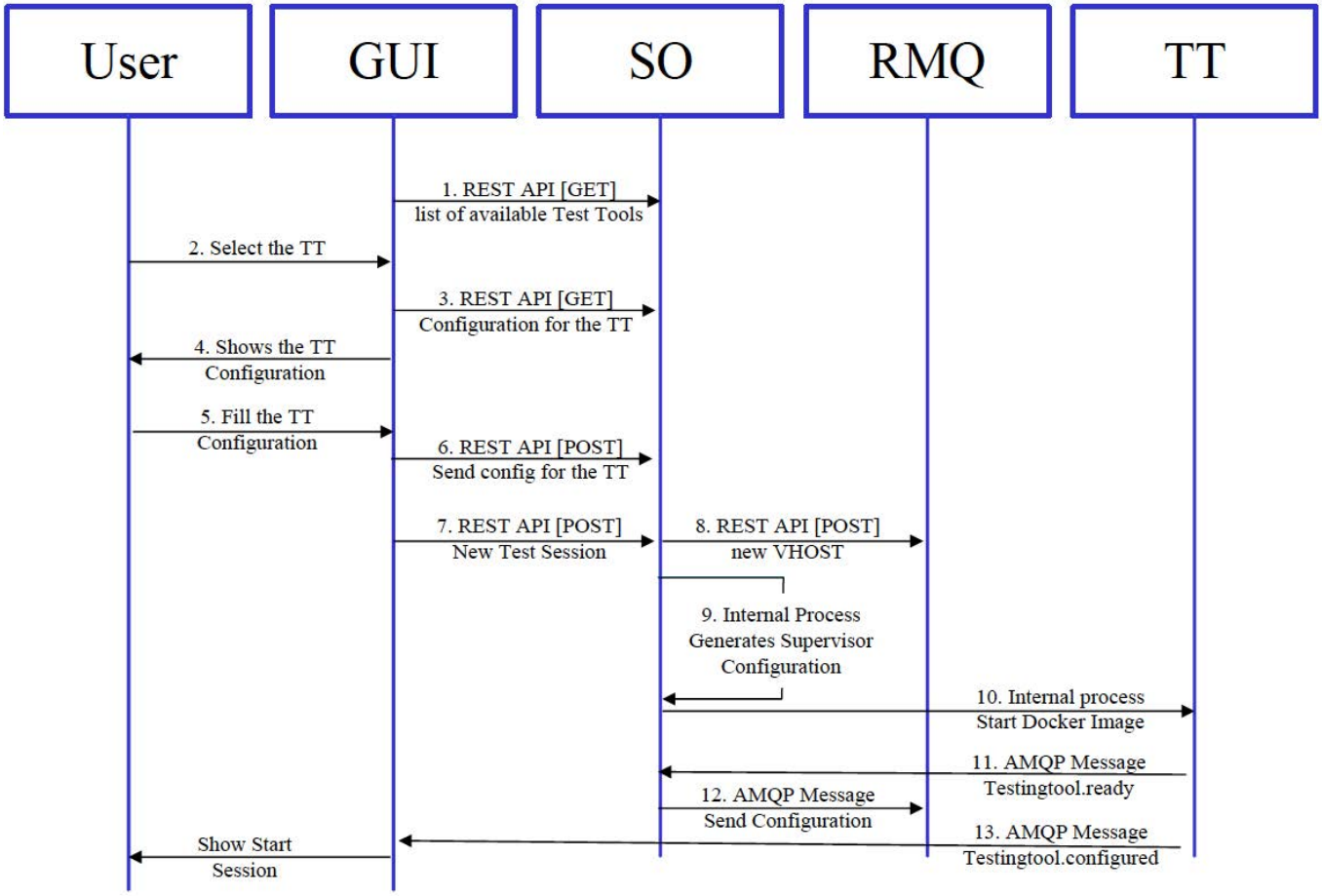

Figure 6 - Orchestration process of a Test Tool

1. The UI queries the SO for the list of available tool types
2. The User selects the desired tool type (in our case - Privacy)
3. The UI queries the SO for the configuration file (index.json).
4. The UI generates the configuration page for the selected tool type using the UI template provided in the configuration file.
5. The User fills in the configuration values (static parameters and the timeline)
6. From the user's input, the UI generates a configuration file specific to the Privacy Test Tool.
7. The UI requests the SO to create a new test session, using SO's REST API. The request includes the configuration file created by the tool's specific UI, which is passed to the SO opaquely.
8. The SO accesses the AMQP broker to create a new vhost for the session and sets the access credentials (generated randomly) and permissions. At this point, the common message bus for the test session is created and further communication between components occurs via AMQP until the session is about to be deleted.

9. The SO reads the Supervisor template (supervisor.conf.j2) provided for the selected tool and generates a Supervisor configuration file sufficient to launch the tool. The AMQP access information is included in this configuration in form of environment variables.
10. The Supervisor daemon running within the F-Interop framework interprets the new configuration file and launches the test tool - in our case, a Docker container is instantiated from the image "testing_tool-privacy". The necessary configuration is passed to the container via environment variables.
11. After the container has been launched, the tools may need some time to fully start up. When this process is complete, the test tool issues a message to the bus indicating its readiness to accept the test configuration.
12. The session orchestrator sends the configuration file from the initial request to create session (Step 7) to the test tool
13. The test tool verifies the configuration, and if it is acceptable, returns a message indicating that the tool is configured and is ready to perform the test. UI displays a button to start the test, accordingly

After these steps, the testing tool is successfully launched, configured and ready for the test execution upon request.

2.2.3 Session Teardown

After this point, the Session Orchestrator does not play an active role during the test execution. Only at the end of the session, the UI can request the teardown of the test session by sending an appropriate REST request to the Orchestrator.

When the session is to be removed, the Session Orchestrator removes the previously created Supervisor configuration, which causes the Supervisor daemon to terminate and remove the Docker container instance. The SO also removes the AMQP vhost, thus terminating the message bus that was used by the session.

2.3 Test Execution Process

Once the session on the Event Bus is created and the Privacy Docker image is executed by the SO, all components are up and running, waiting for the test execution. The interactions among each component involved in the test happen by AMQP messages, mostly defined in the AMQP Libraries.

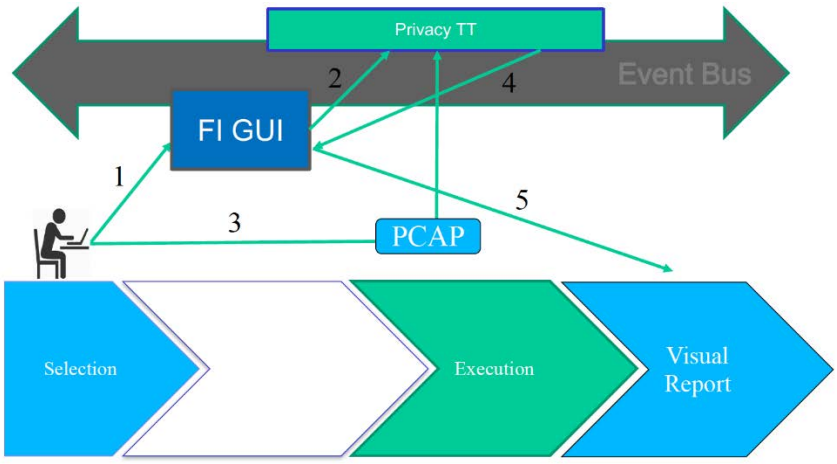

Figure 7 - Privacy Tool Execution Lifecycle

The first release of the Privacy Tool provides two different type of assessments, as shown in Figure 7:

1. The Live assessment
2. The On-Line Assessment.

In a nutshell, the basic difference between these two executions is the way how the PCAP file, the file containing the sniffed packets, is generated.

In the Live Assessment, the PCAP file is automatically generated within the tool. In order to do so, the privacy tool acts as a Man in the Middle (MItM) sniffing the traffic generated by each resource involved in the session. It has the role of a passive observer of the Event Bus traffic, knowing that each packet containing data is encapsulated in an AMQP message by the FI-Agent, the component responsible to seamless connect remote resource to the F-Interop platform.

The capture starts when the START SESSION message is transmitted from the GUI. This message triggers the packet sniffer module running inside the Privacy tool. This is done until the Stop Session message is received. At this point, the communication among resources is considered completed and the packet sniffer can generate a PCAP file containing all those packets received during the session. Such file is sent as body of the Privacy Analyze AMQP message afterward on the Event Bus.

In the On-Line Assessment method, the PCAP file is provided by the user, maybe pre-recorded from a previous test session. Exactly as described in the last step of the Live Assessment, the PCAP file is sent within the Privacy Analyze message as body, and this will trigger the analysis of the privacy tool.

The Privacy tool is able to dissect each packet stored in the PCAP and to extract payloads of the most known IoT application protocols, such as CoAP. On those payloads, pattern matchers are applied to detect possible issues on privacy or personal data leakage.

At the end of such analysis, a report is generated from the tool and is published on the Event Bus as JSON body of the Privacy Analyze Response Message (reply). The JSON content can be further used by any component, for example the GUI or the VizTool, to better visualize the Privacy Analysis result.

2.4 Communication between modules

As showed in Figure 8 - Privacy Message flows, the following components are involved during the execution of the Privacy Test:

1) FI-Agent [1..N]: one for each remote resources involved in the session
2) F-Interop Graphical User Interface (GUI) to start/stop the communication session
3) The Privacy Test Tool in which are running several modules:
   a) The Agent Testing tool
   b) The Packet Dissector
   c) The Sniffer
   d) The Privacy Tool

All these components are connected to the same session in the Event Bus, exchanging AMQP messages to communicate among them.

2.4.1 Generic message format

As described in the API documentation, available here [4], the AMQP message format is the following:

- - - - - - - - - - - - - - - - - - - -

Message routing key: control.session

Message properties: {

"content_type": "application/json",

"message_id": "79263629-1464-4b58-b072-094973e23693",

"timestamp": 1498685266

}

Message body: {

"_api_version": "0.1.32",

"_type": "testingtool.ready",

"description": "Event Testing tool READY to start test suite."

}

- -

The routing key is used to identify the topic of the message. Priorities are used to correlate messages and get details about the packet, such as timestamps, id, etc. All the real information is stored in the Message Body, that MUST contain the _type attributes that define the specificity of the message. Any other attribute can be added in the body to convey any piece of information related to the message type.

2.4.2 Message flow

The Privacy Test Tool has a very simple sequence flow. Since the On-Line Assessment is a subset of the Live Assessment, only the latter is described in this paragraph.

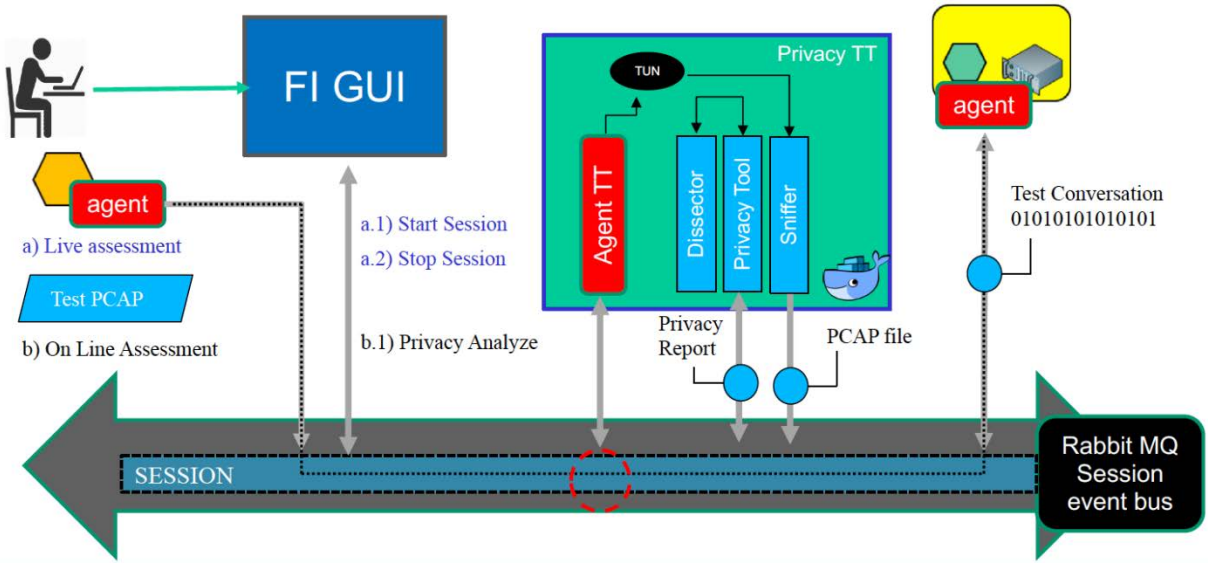

Figure 8 - Privacy Message flows

Figure 8 - Privacy Message flowsshows partially the sequence of messages exchanged during a privacy test session. There are other messages in between that are exchanged by the internal modules of the tool, which are using the Event Bus as well for inter process communication. Here the full list of messages used to perform the privacy analysis ca be found, sorted by execution time:

  - Bootstrap Component
  - Configure Agent
  - Testing Tool Ready
- Start Session (A.1)
  - Sniffer Start
  - Data Packet
- Stop Session (A.2)
  - Sniffer Stop
- Privacy Analyze (B.1)
  - Dissect Packets
- Privacy Analyze Reply (B.1.1)

The component bootstrap is an internal message used to confirm that the module is up and running. The test coordinator waits for those messages to infer the status of the testing tool. If the minimum set of bootstrap messages are not detected, the coordinator might send an error message on the event bus to inform about the privacy test tool anomaly. Here is an example of a Bootstrap Message:

```
- - -
ROUTING_KEY: control.session.bootstrap
PROPS: {
  "message_id": "680f5aee-f08d-46e8-871e-a37e44be8d23",
  "content_type": "application/json",
  "timestamp": 1505986461}
BODY {
        "_api_version": "0.1.30",
        "_type": "testingtool.ready",
        "message": "Testing tool ready to start test suite."
}
- - -
```

AMQP MESSAGE 1 - Bootstrap generic module

All Agents involved in the session MUST be bootstrapped to create the TUN TAP interface on the machine where they are running. Moreover, in the same message some configurations are provided, such as the IPv6 address of the TUN TAP interface, the one used to route packets on the Event Bus. Here an example of the bootstrap agent message:

```
- - -
ROUTING_KEY: control.tun.toAgent.agent_1
PROPS: {…}
BODY: {
        "_type": "tun.start",
        "name": "agent_1",
        "ipv6_prefix": "bbbb",
        "ipv6_host": ":1"
        }
- - -
```

AMQP MESSAGE 2 - Bootstrap Agent

If there are no issues in the configuration process, a **Testing Tool Ready** message is sent on the Event Bus by the Test Coordinator to inform the GUI that the Privacy Test Tool is ready to be used. The message has the following form:

```
- - -
ROUTING_KEY: control.session.bootstrap
PROPS: {...}
BODY: {
        “_type”: “testingtool.privacy.ready”
}
- - -
```

AMQP MESSAGE 3 - Ready Message

To identify a common trigger/start event for each test tool, a **Start Test Suite** message has been defined. This one is usually fired by the GUI, as consequence of a User interaction, notifying to all the components listening on the session, that data packets will comes at any moment.

```
- - -
ROUTING_KEY: control.testcoordination
PROPS: {...}
BODY: {
        “_type”: “testcoordination.testsuite.start”
        }
- - -
```

AMQP MESSAGE 4 - TestSuite Start

As consequence of the previous message, the Test Coordinator (TC) triggers the sniffer component. This can be integrated in the privacy test tool (and it is), or available as a general tool of the session. Anyway, wherever it is located, the **Sniffing Start message** launches the capturing process and all packets (eventually filtered) popping out from a specific interface are stored in a PCAP file. Here an example of the message, which contains the configuration used to instruct the sniffing process:

```
- - -
ROUTING_KEY: control.sniffing.service
PROPS: {...}
BODY {
        "_api_version": "0.1.30",
        “capture_id” : “aa08cdfe-42f3-457d-8e6b-4a8f20a8d0e0”,
        "_type": "sniffing.start",
        “filter_if”: “tun0”,
        “link_id”: “link_1”,
        "filter_proto": "tcp"
}
- - -
```

AMQP MESSAGE 5 - Sniffing Start

As described before, the packets exchanged using the Event Bus session are encapsulated as AMQP message. Such Data Packets looks like as this example below:

```
- - -
ROUTING_KEY: data.tun.fromAgent.agent1
PROPS: {
        "content_encoding": "utf-8",
        "priority": 0,
        "content_type": "application/json",
        "delivery_mode": 2,
        "headers": {}
}
BODY {
        "_type": "packet.sniffed.raw",
        "interface_name": "tun0",
        "data": [
```

```
                96, 0, 0, 0, 0, 56, 0, 1, 254, 128, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 7, 255, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 22, 58, 0, 5, 2, 0,
                0, 1, 0, 143, 0, 108, 229, 0, 0, 0, 2, 4, 0, 0, 0, 255, 5, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 4, 0, 0, 0, 255, 2, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 2
        ]
}
```

AMQP MESSAGE 6 - Packet Sniffed Raw

It is the agent's task to translate an AMQP message to raw PACKET and vice-versa. To interrupt the recording of such data packets, a Stop Message is required. This is the task of the **Test Suite Stop** AMQP Message, that does not kill the session but it triggers the conclusion of some processes and the execution of the post-mortem privacy analysis. Here a simple example of it:

```
- - -
ROUTING_KEY: coordination.testsuite
PROPS: {...}
BODY: {
        "_api_version": "0.1.30",
        "_type": " testcoordination.testsuite.stop",
}
- - -
```

AMQP MESSAGE 7 - TestSuite Stop

Tightly coupled with the previous message, there is the **Sniffing Stop Message**. It is again sent by the TC to kill the data packet capturing process. Before expiring, the sniffer sends a Privacy Analysis request on the Bus, which contains the PCAP file recorded.

Here two examples of the Sniffing Stop followed by the Privacy Analysis AMQP message

```
- - -
ROUTING_KEY: control.sniffing.service
PROPS: {…}
BODY: {
        "_api_version": "0.1.30",
        "_type": "control.sniffing.stop",
}
- - -
```

AMQP MESSAGE 8 - Sniffing Stop

```
- - -
ROUTING_KEY: control.privacy.service
PROPS: {…}
BODY: {
        "_api_version": "0.1.30",
        "_type": "privacy.analyze",
        "filename": "capture_1",
        "file_enc": "pcap_base64",
        "value": " … "
```

}

- - -

AMQP MESSAGE 9 - Privacy Analyze

The privacy test tool uses TSHARK [5] to interpret the PCAP file. This tool is able to “dissect” packets, that means translate what is carried inside the PCAP file and details all protocol stack layers and their specific fields. This task is performed within a specific component that is able to reply to AMQP specific message such as **dissection.dissectcapture** and **dissection.autotriggered.** However, in this first release such process is embedded in the Privacy Tool module.

Once the Privacy Tool terminates the analysis on the dissected packets, a report is generated and it is forwarded to the Event Bus to all those components interested on rendering it or making further computation of it. For instance, this is the task of the GUI and the VizTool. Here an example of the reply message, without the completed JSON value, to make the message more readable.

- - -

ROUTING_KEY: control.privacy.service.reply

PROPS: { …}

BODY: {

"_api_version": "0.1.30",

"_type": "privacy.analyze.reply",

"ok": "true",

"verdict": "<JSON content>",

}

- - -

AMQP MESSAGE 10 - Privacy Message (REPLY)

2.5 Test Tool Configuration

In this first implementation, there is no configuration required. The tool uses all Data Matchers available and provides a report at the end of the analysis. The only thing the user should select on the GUI is the type of assessment to perform, Live or On-Line. From the perspective of the Privacy Tool, the behaviour doesn’t change since the PCAP file is provided by the same AMQP message. As described before, what changes from these two modalities is how packets are captured, and therefore who records the PCAP file.

3 Privacy Test Tool Modules Functionality

This section describes in more detail the individual components of the Privacy Test Tool. As showed in Figure 8, the Privacy Docker Container hosts five different modules:

1. The Test Coordinator (TC)
2. An F-Interop Agent (FI-Agent)
3. The Packet Sniffer (PS)
4. The Packer Dissector (PD)
5. The Privacy Tools (PT)

The Privacy Container uses Supervisor [6], a Process Control System that spawns all processes and monitor them during the execution of a test. Supervisor uses a configuration file to list the required processes to execute and write the related logging files. The used configuration file is described in Appendix 6.1.3

The source code of those modules, except for the Agent, is available under the Privacy_testing_tool.git project, in the F-Interop code repository [7]. The agent code is available at [8]

In the following sections of this chapter, there is a general description of each modules involved.

3.1 Test Coordinator (TC)

The Test Coordinator (TC) module is responsible for the Privacy Test Tool lifecycle. It glues everything together, verifying that all internal components are working properly, triggering and configuring them by exchanging specific AMQP messages. TC also reports internal issues and, as a future improvement, it will be able to hot deploy configurations and restart tasks on demand.

In a nutshell:

- It manages the lifecycle of the overall Privacy Test Tool
- It checks component status
- It is connected to the Event Bus
- It sends bootstrap, configuration and synchronization messages
- No configuration needed in the first release
- It does not provide any statistics

3.2 F-Interop Agent TT

The “Agent” is a python module that allows FI-User to connect IUTs to the F-Interop server. Communication between the agent and the server is authenticated and secure, using the Event Bus Session credential.

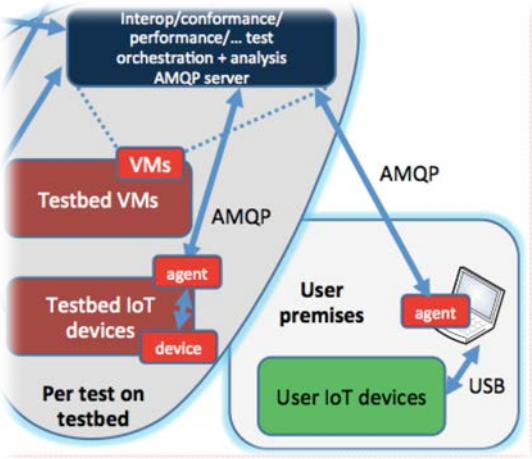

Figure 9 - F-Interop Agent

Through the agent, as showed in Figure 9, the F-Interop server can (remotely) interact with the IUT, for example by changing its configuration or injecting packets in the link/interface the IUT is listening to. Similarly, the agent reports to F-Interop the data packets sent from the IUT.

The agent creates a virtual interface on the machine where it is executed, called TUN TAP [9]. The agent main task is to route packets sent to a specific subnet (usually IPv6 bbbb subnet), that is basically connected to an F-Interop service, through the Event Bus and vice-versa.

This transfer from the network to the Event Bus is done by encapsulating raw packet into a AMPQ Message as described in the example AMQP MESSAGE 6 - Packet Sniffed Raw

In this way, the packet routing to the right destination is handled by the application level, usually defined in the session configuration. A Packer Router component should be present in the service to take care the right delivery of a packet to the right agent connected to the Event Bus.

AMQP MESSAGE 2 - Bootstrap Agent provides an example of configuration message.

The agent, which is running in the Privacy Test Tool container, reads all Data Packets exchanged within the session. This is possible through a special packet router process that forwards messages from IUT agents to the privacy agent (Agent_TT).

The agent is also responsible to extract packets from the Event Bus and forwarded them in the TUN TAP interface. This interface, created in the Docker container, is monitored by the Packer Sniffer, as described in the next section.

3.3 Packet Sniffer

The Packet Sniffer (PS) is the process in charge of capturing packets. This task is done by using TCPDUMP [10], one of the most famous Linux Tool largely used to debug network traffic.

The python module that implements the PS establishes a connection with the test session in the Event Bus, waiting for incoming commands, such as the Start of the Sniffing capture or the Stop message.

The start command carries a number of parameters used to instruct the TCPDUMP process. Examples for both START and STOP Sniffing Message are provided in section 2.4.2.

Table 2 - Start Sniffing Body Parameters

Parameters	Value	Description
Routing Key (RK)	control.sniffing.service	AMQP message Routing Key
_type	sniffing.start	AMQP message type
filter_if	tun0	The interface name to listen
api_version	0.1.30	API version of the AMQP Message Library
filter_proto	TCP	Packet filter to apply during the capturing session
capture_id	9fe5b09e-d0ee-43bb	A unique ID to identity the capture session

Table 2 describes all parameters provided within the START Message; using them, the python process starts a TCPDUMP background OS process using the following syntax:

*tcpdump -K -i **tun0** -s 200 -U -w **capture_id_9fe5b09e-d0ee-43bb TCP** &*

Such background process is killed when the STOP message is received.

The PS can automatically send the Privacy Analyze message request on the Event Bus, providing the recorded PCAP file as message body.

Moreover, all captures created by the PS during a session are stored in a local repository. A specific PCAP file can be retrieved by using the GET CAPTURE AMQP message request, which contains the **capture_id** parameter as described in the following example:

```
- - -
ROUTING_KEY: control.sniffing.service
PROPS: {...}
BODY {
        "_api_version": "0.1.30",
        “capture_id” : “aa08cdfe-42f3-457d-8e6b-4a8f20a8d0e0”,
        "_type": "sniffing.getcapture",
}
- - -
```

AMQP MESSAGE 11 - Get Sniffing Capture

3.4 Packet Dissector

The **P**acket **CAP**ture (PCAP) format is used by TCPDUMP to record network traffic. In this file, each captured packet contains the full protocols stack of the network frame.

In order to analyse privacy issues inside IoT protocols and to simplify the extraction of their payloads, an additional tool has been introduced. This tool is called TSHARK [5], which is similar to TCPDUMP as a network packet analyser, but with some more functionalities. Besides capturing packets, this tool can translate PCAP files into other formats, such as JSON file.

Considering such similarities of both Packet Dissector (PD) and Packet Sniffer (PS), these two modules might be “merged” in the future into a unique component. However, in the first iteration the Privacy Test Tool keep these two modules separated.

Focusing on the PD task, the conversion from PCAP to JSON has a twofold beneficial effect. The first one enables an easier and efficient way to access packet fields, by using well-known JSON libraries available in different programming languages. Secondly, it translates each layer of the frame using specialized component called “dissectors”. These dissectors are plugins of the TSHARK tool, with the scope of converting raw data into specific protocol layer fields, such as headers, flags, payloads, etc.

TSHARK comes already with CoAP and MQTT dissectors. This means that the tool is ready to understand most of widely used application protocols in the IoT world. In the Privacy Test Tool container, TSHARK is cloned and installed during the compilation of the Docker image. To enable the JSON translation it is mandatory to use the branch version 2.2.0

```
RUN apt-get install -y git
RUN git clone --branch wireshark-2.2.0 --depth=1
https://github.com/wireshark/wireshark
RUN cd wireshark; ./autogen.sh
RUN cd wireshark; ./configure --disable-wireshark --disable-warnings-as-errors
RUN cd wireshark; make
RUN cd wireshark; make install
RUN ldconfig
RUN tshark -v
```

The Process Dissector is a Python function that receives the file path to be converted and it returns the path of the JSON file. The conversion is done by using the following command:

*tshark -r **pcap_file_path** -l -n -x -T json > json_file_name*

The JSON file usually keeps the original file name, but with a different extension (.json). Both the PCAP and the JSON file are stored in a local repository.

3.5 Privacy Tool

The Privacy Tool (PT) is the heart of the Privacy Test Tool. It receives in input the JSON file containing an IoT conversation and, after extracting the full list of packet payloads, it applies rules to match privacy patterns on each of them. While performing such task, the PT generates a report containing statistics about the IoT conversation, such as the number of involved devices, amount of exchanged data (in bytes), protocols and more. In addition, for each detected privacy issue, an entry is added with the following information:

- **Frame ID** of the packet in which the issue has been detected
- **Protocol** of the payload
- **Token**, the text string of the pattern detected.
- **Privacy TAG**, the semantic meaning of the issue

The report contains also the frames of those packets containing at least on privacy issue.

In order to define privacy patterns, PT uses Data Matchers, a specialized set of components which define a Privacy pattern and the related method to detect it (see Figure 10). This makes the tool flexible and modular in a way that more and more patterns can be added or tuned progressively.

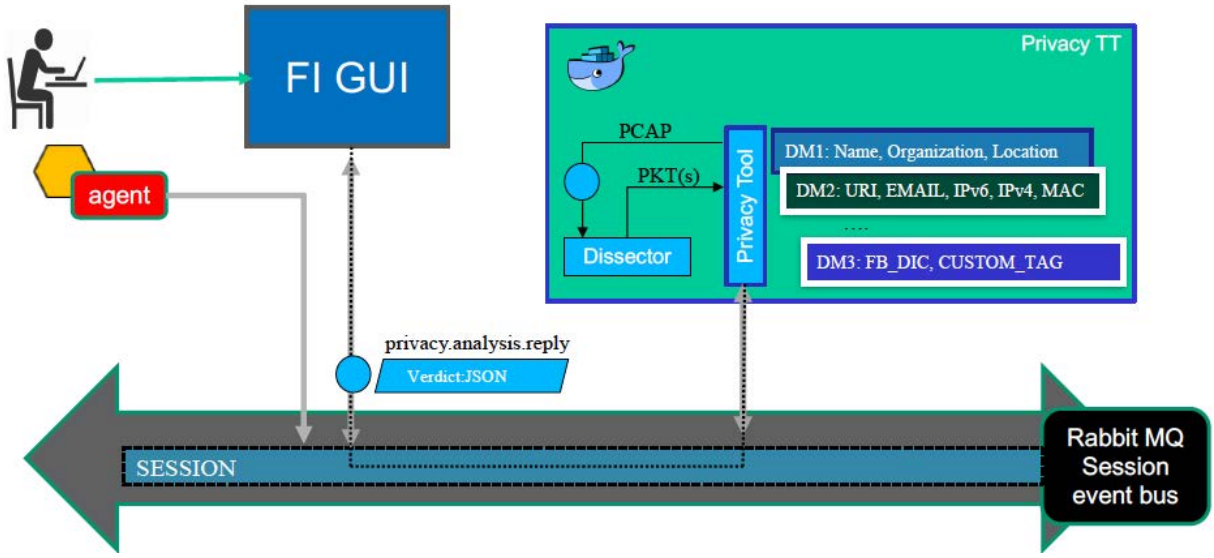

Figure 10 - Privacy Tool Data Matcher

In detail, a Data Marcher (DM) is a standalone module/file that contains all the business logic to detect a specific privacy issue. The PT loads DMs from a specific folder, and it starts forwarding payloads to such matchers. Each DM filters and analyses the content of the payload and looks for some patterns. As output, each DM provides a list of <TOKEN, TAG> pairs detected in that specific payload.

The Privacy Tool takes care of assembling those outputs and creates a unique final report. (See Annex 6.3)

In the first release of the tool, there are three DMs available:

- Organization and Naming Data Matcher
  - **TAGs**: ORGANIZATION, NAME, LOCATION
  - It uses Natural Processing Languages (NPL) to detect such tokens
- Regular Expression Data Matcher
  - **TAGs**: URI, IPv6, IPv4, EMAIL, URI, IDs, GEO Coordinates
  - It uses Specialized Regular Expression to detect such tokens
- Forbidden Dictionary Data Matcher
  - **TAGs**: FORBIDDEN_KEY
  - It uses a dictionary to detect such tokens. The dictionary can be configured.

4 Future Developments

The first release of the Privacy Test Tool has been completed and demonstrated during the formal review in Brussels. The modules described in this document have been implemented, as well as their functionalities.

As future work, some new improvements of those already available functionalities will be integrated, such as:

1. Send Partial Report on the fly
2. Cascade integration with other testing tools
3. Custom Privacy Policy at configuration time
4. Integration of new Protocols (MQTT)
5. Add other Data Matchers
6. Bug Fixing & Improvements

Send Partial Report is about providing information while a conversation is executed on the Event Bus. This means changing the way on how packets are checked, informing the user with an alert whenever an issue is detected during a test session.

Cascade integration is the idea of having the Privacy Test Tool as a platform functionality, adding it in cascade to any other test running in the F-Interop Platform. For instance, besides doing a CoAP Conformance Test, a Privacy Test on the content exchanged can be performed as well.

Custom Privacy Policy is a way to configure the Privacy Test Tool before deploying it. During this stage, the user might be able to provide several configurations, such as specific forbidden keywords, regular expression to check specific patterns, and more.

New Protocols. So far, the Privacy Test Tool only checks CoAP Payloads. At least the other most used protocol, MQTT, should be integrated in the tool.

More Data Matchers might be integrated in the Core of the Privacy Test to enhance the detection of Privacy and Security issue, according to the experience and the new requirements of the F-Interop users.

Bug Fixing & Improvements will be also applied to the solution, making it more stable and following the F-Interop internal changes.

These are possible improvements of the Privacy Test Tool provided at the first iteration. Other developments might be done within the Task, in order to prevent attacks of malicious users who could passively observe patterns of encrypted communications.

5 References

[1] **Session Orchestrator** (SO) Rest API - http://doc.f-interop.eu/#session-orchestrator

[2] **Docker Container** - https://www.docker.com/what-container

[4] F-Interop **API Document** - http://doc.f-interop.eu

[3] Privacy Test Tool **GIT Repository** - https://gitlab.distantaccess.com/f-interop/privacy_testing_tool.git

[6] **Supervisor** - http://supervisord.org

[10] **TCPDUMP**, A Network Sniffer - http://www.tcpdump.org

[9] **TUN TAP** Interface - http://www.naturalborncoder.com/virtualization/2014/10/17/understanding-tun-tap-interfaces/

[5] **TSHARK** Network analyser - https://www.wireshark.org

[8] **Agent** GIT Repository - https://gitlab.distantaccess.com/f-interop-contributors/agent

6 Annex

This annex contains some of the configuration used to execute the test tools plus some other details useful to understand the execution of the tool

6.1 Configuration Files

6.1.1 Privacy Test tool Dockerfile

```
# Privacy Test Tool Container
FROM ubuntu:16.04
MAINTAINER luca.lamorte@uni.lu

RUN apt-get clean
RUN apt-get update
RUN apt-get upgrade -y
RUN apt-get update -y -qq && apt-get -y -qq install python3-dev
RUN apt-get -y install python3-setuptools
RUN apt-get install -y apt-utils
RUN     apt-get -y install python3-pip
RUN     apt-get -y install python-pip
RUN     apt-get -y install supervisor
RUN     apt-get -y install tcpdump
## TSHARK INSTALL
RUN apt-get install -y automake autoconf
RUN apt-get install -y libgtk2.0-dev libglib2.0-dev libpcap0.8-dev
RUN apt-get install -y flex bison libtool-bin libgcrypt-dev
## CLONE REPOS
RUN apt-get install -y git
RUN git clone --branch wireshark-2.2.0 --depth=1
https://github.com/wireshark/wireshark
RUN cd wireshark;./autogen.sh
RUN cd wireshark;./configure --disable-wireshark --disable-warnings-as-
errors
RUN cd wireshark; make
RUN cd wireshark; make install
RUN ldconfig
RUN tshark –v
## ADD the DIRECTORY of the TOOL
ADD . /privacy_testing_tool
ENV PATH="/privacy_testing_tool:$PATH"
WORKDIR /privacy_testing_tool

#py2 requirements
RUN pip install -r privacy_testing_tool/agent/requirements.txt
#py3 requirements
RUN pip3 install -r privacy_testing_tool/test_coordinator/requirements.txt
RUN pip3 install -r privacy_testing_tool/packet_router/requirements.txt
RUN pip3 install -r privacy_testing_tool/sniffer/requirements.txt
RUN pip3 install -r privacy_testing_tool/privacy_tool/requirements.txt
## JAVA RUNTIME ENVIRONMENT
RUN     apt-get -y install default-jre
## API PORT
EXPOSE 8181 8181
# launch processes
CMD supervisord -c supervisor.conf
```

6.1.2 Session Orchestrator configuration files

Index.json


```
{
  "_type": "testsuite.manifest",
  "testing_tool":"Privacy Testing Tool",
  "version":"0.0.1",
  "test_type": "privacy",
  "ansible_playbook": "ansible/main.yml",
  "owner": "F-Interop",
  "mantainer": "Luca Lamorte",
  "mantainer_email": "luca.lamorte@uni.lu",
  "protocols_under_test": ["CoAP"],
  "underlying_supported_protocol": ["ipv6","udp"],
  "agent_names": ["privacy_agent", "coap_server_agent", "coap_client_agent"],
  "iut_roles": ["coap_client", "coap_server"],
  "available_location_models:": [],
  "ui_information":[
    {
      "field_name": "Testing Tool Description",
      "type": "text",
      "value": ["Privacy Testing Tool performs privacy analysis of the data exchanged while running different kind of tests","The tool checks any possible privacy and personal data leakage, providing a full report of the conversation"]
    },
    {
      "field_name": "Testing tool version",
      "type": "selection",
      "values": ["0.0.1"],
      "mandatory": false
    },
    {
      "field_name": "Test cases selection",
      "type": "multi-selection",
      "values": [
        "OnLine Privacy assessment",
        "Live Privacy assessment"
      ],
      "mandatory": false
    }
  ]
}
```

supervisor.conf.j2

```
-------
[program:{{ session }}|testing_tool]
stopsignal=TERM
killasgroup=true
autostart=false
stdout_logfile = %(here)s/logs/{{ session }}-testing_tool-stdout.log
stderr_logfile = %(here)s/logs/{{ session }}-testing_tool-stderr.log
command = docker run
   --env AMQP_URL={{ amqp_url }}
   --env AMQP_EXCHANGE={{ amqp_exchange }}
   --rm
   --privileged=true
   --sysctl net.ipv6.conf.all.disable_ipv6=0
   --name="session_{{ session }}-testing_tool-privacy"
   testing_tool_privacy
   supervisord --nodaemon --configuration supervisor.conf
```

6.1.3 Supervisor configuration file for the Privacy Test Tool container

supervisor.conf

```
[unix_http_server]
file=/tmp/supervisor.sock   ; (the path to the socket file)

[supervisord]
logfile=/tmp/supervisord.log ; (main log file;default $CWD/supervisord.log)
logfile_maxbytes=50MB        ; (max main logfile bytes b4 rotation;default
50MB)
logfile_backups=10           ; (num of main logfile rotation
backups;default 10)
loglevel=info                ; (log level;default info; others:
debug,warn,trace)
pidfile=/tmp/supervisord.pid ; (supervisord pidfile;default
supervisord.pid)
nodaemon=false               ; (start in foreground if true;default false)
minfds=1024                  ; (min. avail startup file descriptors;default
1024)
minprocs=200                 ; (min. avail process descriptors;default 200)

[rpcinterface:supervisor]
supervisor.rpcinterface_factory =
supervisor.rpcinterface:make_main_rpcinterface

[supervisorctl]
serverurl=unix:///tmp/supervisor.sock ; use a unix:// URL  for a unix
socket

[program:tc]
command = python3 -m privacy_testing_tool.test_coordinator
autorestart=false
stopsignal=INT
stopasgroup=true
loglevel=debug
```

```
redirect_stderr=true
stdout_logfile = /var/log/test_coordinator-stdout.log
stdout_logfile_maxbytes = 10MB
stdout_logfile_backups = 5

[program:packet-sniffer]
command = sh -c "sleep 1;python3 -m
privacy_testing_tool.sniffer.packet_sniffer"
user=root
stopsignal=INT
stopasgroup=true
autorestart=false
redirect_stderr=true
loglevel=info
stdout_logfile = /var/log/packet_sniffer-stdout.log
stdout_logfile_maxbytes = 10MB
stdout_logfile_backups = 5

[program:privacy]
directory= ./privacy_testing_tool/privacy_tool
command = sh -c "python3 -m pbd"
autorestart=false
stopsignal=INT
stopasgroup=true
loglevel=debug
redirect_stderr=true
stdout_logfile = /var/log/privacy-stdout.log
stdout_logfile_maxbytes = 10MB
stdout_logfile_backups = 5

[program:agent]
directory= ./privacy_testing_tool/agent/
command = sh -c "sleep 1;python -m agent connect --url %(ENV_AMQP_URL)s --
name privacy_agent"
user=root
stopsignal=INT
stopasgroup=true
autorestart=false
loglevel=debug
redirect_stderr=true
stdout_logfile = /var/log/agent-stdout.log
stdout_logfile_maxbytes = 10MB
stdout_logfile_backups = 5

[program:bootstrap-agent-TT]
command = sh -c "sleep 4; python3 -m
privacy_testing_tool.agent.utils.bootstrap_agent privacy_agent bbbb :7"
user=root
stopsignal=INT
stopasgroup=true
autorestart=false
redirect_stderr=true
loglevel=info
stdout_logfile = /var/log/packet_sniffer-stdout.log
stdout_logfile_maxbytes = 10MB
stdout_logfile_backups = 5

[program:packet-router]
command = sh -c "sleep 2; python3 -m
privacy_testing_tool.packet_router.packet_router"
autorestart=false
```

```
stopsignal=INT
stopasgroup=true
loglevel=debug
redirect_stderr=true
stdout_logfile = /var/log/packet_router-stdout.log
stdout_logfile_maxbytes = 10MB
stdout_logfile_backups = 5



[program:api]
directory= ./privacy_testing_tool/privacy_tool
command = sh -c "python3 -m pbd.api"
autorestart=false
stopsignal=INT
stopasgroup=true
loglevel=debug
redirect_stderr=true
stdout_logfile = /var/log/pbd_api-stdout.log
stdout_logfile_maxbytes = 10MB
stdout_logfile_backups = 5
```

6.2 Example of a Privacy Report (JSON)


```
{
  "session_id": "bb5a2968-76e8-4650-baa4-c3d2803b8427",
  "protocols": [
    "eth:ethertype:ip:udp:coap",
    "eth:ethertype:ip:udp:coap:data-text-lines"
  ],
  "is_final": true,
  "type": "Anomalies Report",
  "num_packets": 84,
  "version": "0.0.4",
  "anomalies": 16,
  "byte_exchanged": 10704.0,
  "status": "positive",
  "conversation": [
    {
      "num_anomalies": 0,
      "anomalies": [],
      "id": "10.41.7.222",
      "protocols": [
        "eth:ethertype:ip:udp:coap"
      ],
      "size": 2664.0,
      "packets": 42
    },
    {
      "num_anomalies": 16,
      "anomalies": [
        {
          "privacy_tag": "ORGANIZATION",
          "pkt_id": "66",
          "device": "10.186.24.12",
          "timestamp": "Jun 30, 2017 13:59:59.189304000 UTC",
          "protocol": "eth:ethertype:ip:udp:coap:data-text-lines",
          "detected_token": "uni.lu"
        },
        {
          "privacy_tag": "PERSON",
          "pkt_id": "66",
          "device": "10.186.24.12",
          "timestamp": "Jun 30, 2017 13:59:59.189304000 UTC",
          "protocol": "eth:ethertype:ip:udp:coap:data-text-lines",
          "detected_token": "Luca"
        },
```

```
  …
  {
    "privacy_tag": "FORBIDDEN_KEYWORD",
    "pkt_id": "76",
    "device": "10.186.24.12",
    "timestamp": "Jun 30, 2017 14:00:08.577606000 UTC",
    "protocol": "eth:ethertype:ip:udp:coap:data-text-lines",
    "detected_token": "F-Interop"
  },
  {
    "privacy_tag": "IPv4",
    "pkt_id": "70",
    "device": "10.186.24.12",
    "timestamp": "Jun 30, 2017 14:00:02.569730000 UTC",
    "protocol": "eth:ethertype:ip:udp:coap:data-text-lines",
    "detected_token": "10.2.2.101"
  },
  {
    "privacy_tag": "EMAIL",
    "pkt_id": "66",
    "device": "10.186.24.12",
    "timestamp": "Jun 30, 2017 13:59:59.189304000 UTC",
    "protocol": "eth:ethertype:ip:udp:coap:data-text-lines",
    "detected_token": "luca.lamorte@uni.lu"
  },
  {
    "privacy_tag": "IPv6",
    "pkt_id": "68",
    "device": "10.186.24.12",
    "timestamp": "Jun 30, 2017 14:00:00.764880000 UTC",
    "protocol": "eth:ethertype:ip:udp:coap:data-text-lines",
    "detected_token": "fe80::20c:29ff:fed2:1298"
  },
  {
    "privacy_tag": "URI",
    "pkt_id": "72",
    "device": "10.186.24.12",
    "timestamp": "Jun 30, 2017 14:00:03.971695000 UTC",
    "protocol": "eth:ethertype:ip:udp:coap:data-text-lines",
    "detected_token": "http://finterop.eu"
  }
],
"id": "10.186.24.12",
"protocols": [
  "eth:ethertype:ip:udp:coap",
  "eth:ethertype:ip:udp:coap:data-text-lines"
```

```
      ],
      "size": 8040.0,
      "packets": 42
    }
  ],
  "timestamp": 1505821352.2997234,
  "testing_tool": "Privacy Testing Tool",
  "_type": "pcap_file"
}
```

6.3 Example of a Test Session

This example shows the full execution of a Privacy Test Session.

Step1: In the GUI, the FI-User selects the Privacy Test among different testing tools. In the first iteration, there is no need to provide any sort of configuration. The only action is to select which one of the two assessments should be executed. Once this is done, the GUI starts the session and executes an instance of the Privacy Tool by using the SO Rest API(s).

At the end of this step all components of the Privacy Test Tool are running and they are connected to the new session created for the test. Then the user can use the GUI to officially start the session with the according message (*testcoordination.testsuite.start*) or uploading the PCAP file that will be sent in the *privacy.analyze* message and further analysed by the tool.

STEP 2: After the PCAP file is sent by the user or captured during an IoT Test, the Privacy Test Tool analyses the contained packets and generates a verdict. The verdict is sent back as JSON body in the *privacy.analysis.reply* message (see Figure 11).

The JSON content is used by the GUI to create the visual report showed in Figure 12. More details about the visualization part are provided in Deliverable D3.4.

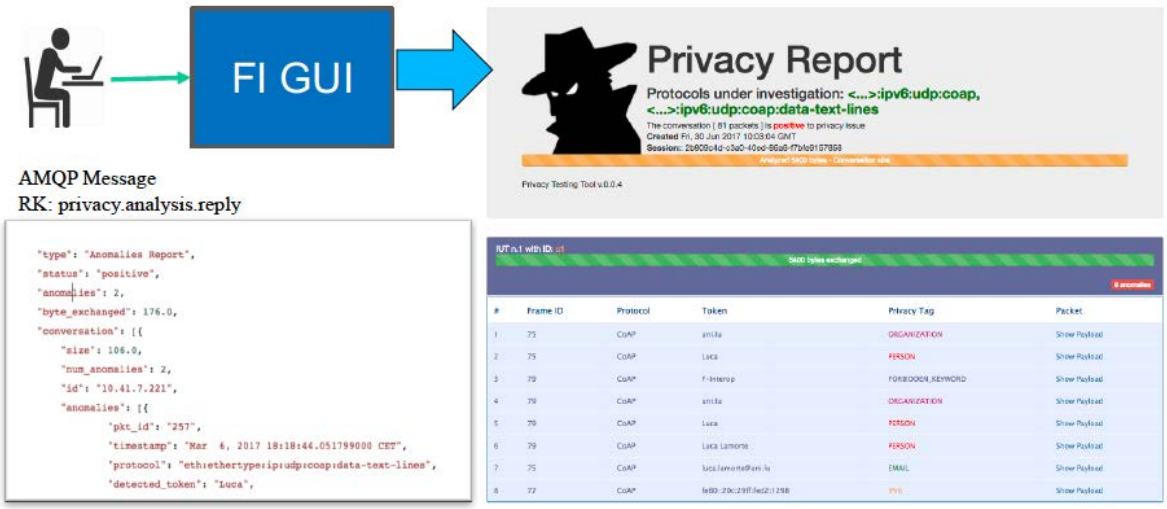

Figure 11 - Example of privacy analysis reply

The report starts with a header that provides a general **overview** of the privacy analysis, such as protocol, time, number of packets exchanged and more. A positive status means that at least one issue has been detected. In the second part of the report for each device involved in the conversation there is a section that contains the amount of data transmitted and the list of privacy issues detected. An issue is described as the entry of a table. Each entry contains the Frame ID of the packet involved, the protocol used to transfer the data, the token detected and a Privacy TAG, that defines a category of the threat. Such category provides the “Semantical” description of the privacy leak detected. To get more details, it is possible to click on the show payload button, which provides the full payload of that packet.

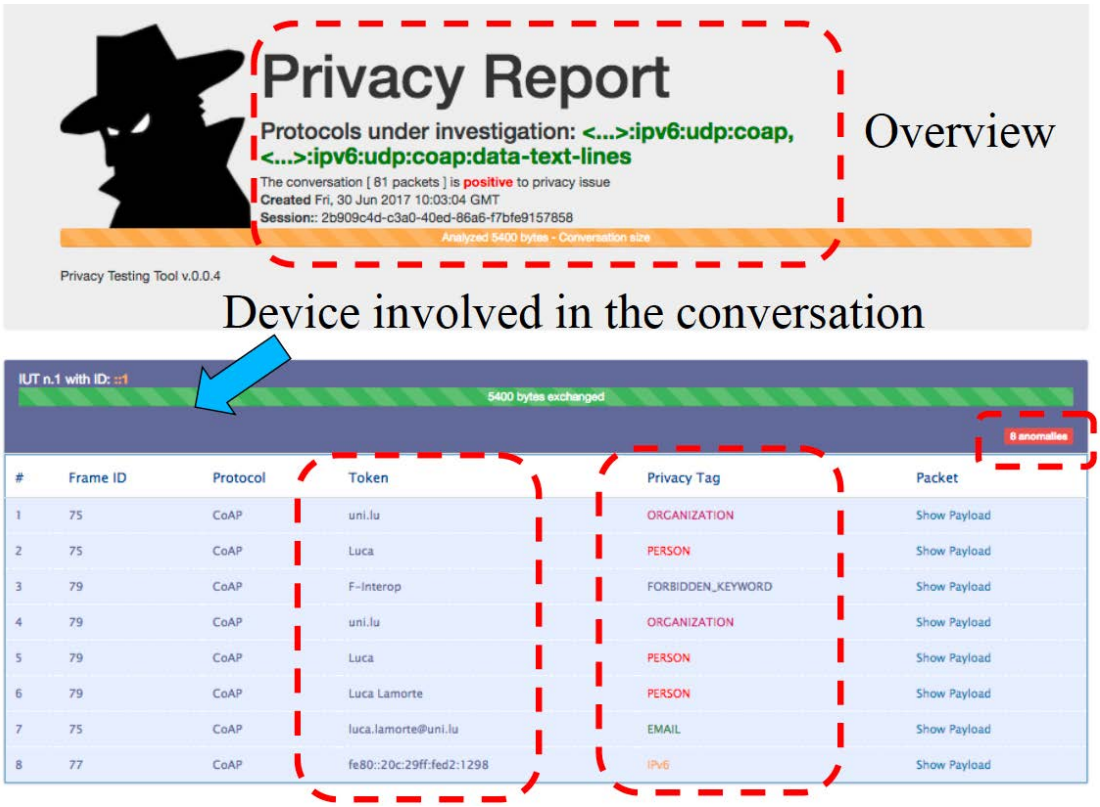

Figure 12 - Section of the visual Privacy Report