
# Vulnerability and Attack Repository for IoT

*Activity 6*

## Dissemination & Endorsement

*Deliverable 6.4*

## Lessons Learned Report

**Co-financed by the Connecting Europe Facility of the European Union**

## Deliverable Information

| Document Administrative Information | |
|---|---|
| Project Acronym: | VARIoT |
| Project Number: | 2018-EU-IA-0100 |
| Deliverable Number: | 6.4 |
| Deliverable Full Title: | Lessons Learned Report |
| Deliverable Short Title: | Lessons Learned Report |
| Document Identifier: | VARIoT-64-LessonsLearnedReport-final |
| Beneficiary in Charge: | IMT |
| Report Version: | 1.0 |
| Contractual Date: | 31/10/2022 |
| Report Submission Date: | 31/10/2022 |
| Dissemination Level: | PU |
| Nature: | Report |
| Lead Author(s): | IMT |
| Co-author(s): | Mondragon University, NASK, Shadowserver, SMILE |
| Keywords: | [List of free keywords relevant to the deliverable] |
| Status: | __ draft, _x_ final, __ submitted |

## Change Log

| Date | Version | Author/Editor | Summary of Changes Made |
|---|---|---|---|
| 31/10/2022 | v1.0 | IMT | Final report submission |
| | | | |

# Table of Contents

# List of Abbreviations

# Executive Summary

VARIoT is a CEF Telecom call project that contributes to the construction of European data platforms. In particular, this project deals with the collection, analysis and sharing of a large amount of heterogeneous data originating from various sources relevant to vulnerabilities and attacks against IoT devices. The main outcomes of the project include a database of information on vulnerabilties and exploits of IoT devices, mechanisms of correlation of various types of information, a vulnerability information search engine, a catalogue of IoT related malware, a set of scanning reports of exposed IoT devices, a system for IoT devices network anomaly detection, a MISP implementation of the open data, and aggregated and anonymised statistics of infected and vulnerable IoT devices, and a number of both legitimate and compromised IoT device traffic datasets.

In order to promote the project and disseminate its achievements, we are also providing a collective feedback of our experiences collecting, analysing and sharing cybersecurity-related IoT data. This *Lessons Learned* report aims at providing guidance on effective development of similar services and recommendations on efficient approaches to the collection and harmonization of fragmented, distributed IoT data. It is divided in two main parts: the first part deals with partner-specific approaches to eliciting the generation of, collecting, correlating, analyzing and sharing cybersecurity-related IoT data of diverse natures, be it vulnerabilities, exploits, malware, traffic or indicators of compromise (IoCs).

**In the first part,** the first section is dedicated to the collection and fusing of heterogeneous sources of IoT data dealing with vulnerabilities and exploits. It is a 2-step process that first accommodates raw data from numerous vulnerability databases around the world, with the goal to harmonize the descriptions of vulnerabilities through a standardization step resulting in a unified database, termed *low database*. The second step combines information from low databases into *medium databases* that correlate entries that share a given CVE ID. The feedback focuses on approaches and algorithms used to produce these different databases. Finally, the first section concludes on how an ELK stack is leveraged to automate the update of the different databases.

A second section deals with various feedbacks on producing large-scale and efficient scan campaigns to detect vulnerable, exposed IoT devices. This section focuses on hard problems of scanning IoT devices at Internet scale, namely scanning IPv6 addresses and fingerprinting IoT devices. Both problems are seldom addressed, in particularly IPv6 scanning is considered almost impossible given the number of theoretical addresses. This was circumvented by relying on particular hitlists of already observed IPv6 addresses gathered from a variety of sources. It remains an open problem as existing tools are still immature. On the other hand, fingerprinting IoT devices is challenging as it attempts to uncover the make and model of every accessible device. Again, existing scanning tools are not scalable, and the problem was addressed mainly by data-driven pattern discovery from data collected in other scans, in particular HTTP scans. It actually yielded another surprising discovery that port forwarding is quite common: different devices hidden behind a single IP address would reply to different scans on different ports or services.

A third section on IoT traffic capture highlights points of attention with respect to setting up an IoT traffic testbed for the purpose of collecting legitimate IoT traffic that could be used later for fingerpriting or anomaly detection. It details how to select equipment with respect to the set of devices to be tested, the configuration of the collection device network-wise but as well in terms of throughput and storage. Not surprisingly, a micro-computer can be advantageously

leveraged to collect live traffic from a smart home sized environment.

In the fourth section, the analysis of IoT malware is detailed, especially how honeypots need to be set up in order to effectively be able to capture IoT attacks. The malware itself required the building of an entire overlay system on top of a new proprietary-developed honeypot. Finally, it is also mentioned how sinkholing IoT DGA malware was performed. The final section of the first part focuses on the publication and sharing of IoT data collected during the course of the IoT project through the Open Data portals, both at domestic and European levels. The insights cover the data formats, the contact procedures, and the quality assessment.

**In the second part,** a first section deals with the processing of large volumes of data at the network level. Tools and specific command lines are shared to perform packet level processing tasks. A few pitfalls are also discussed.

A second section discusses storage issues for packet captures. General packet capture tips are presented with respect to complete payload capture and timestamp alignment. Best practises are also shared to ensure a reliable packet capture. Then, an example tool chain is described to perform cloud-based cluster storage of packet captures.

Finally, privacy is addressed in a third section. It covers different types of data, including personally identifiable information or not. In cases where GDPR must be strictly observed, it is often required to start with a privacy impact analysis and contact your institution's data protection officer as early as possible, preferably before collecting any data. When human subjects are involved, collection consent must be sought and data retention must have an explicit end date.

It is hoped that the feedback collected in this *Lessons Learned* report will find its way to practitioners around Europe and enable the reproduction of results obtained in the VARIoT project. Such contents are also fuel for discussion among data stakeholders that definitely would benefit from the parts of this report dedicated to sharing the data.

# 1 Introduction

## 1.1 Purpose and Scope of the Document

This *Lessons Learned* report is one of the main media that contributes to the dissemination of the VARIoT project results, increasing the awareness among the public and relevant cybersecurity stakeholders about the vulnerability and security of IoT. In particular, its purpose is to convey the experience gathered during the course of the project, providing guidance on effective development of similar services and recommendations on efficient approaches to collection and harmonization of fragmented, distributed data related to cybersecurity (especially in IoT).

## 1.2 Structure of the Document

This document is organised as follows: Section 1 provides information about the report content while the technical contents are described in two distinct sections: one deals with specific contributions or services developed by partners for collecting, storing and sharing the IoT data (Section 2); the other deals with more general technical considerations on processing large amount of cybersecurity related data (Section 3). The deliverable is concluded in Section 4.

# 2 IoT Data Collection and Sharing

## 2.1 Collect and fuse heterogeneous sources of IoT data

### 2.1.1 Building Raw and Low databases

Acquiring data from publicly available sources proved to be a constant challenge through the project development. Most of the sources do not provide a dedicated way to get their data in an automatic way, only national vulnerability databases provide data feeds. Even for them, using data feeds is usually a compromise: American NVD and Japanese JVNDB feeds miss some minor information fields which are available on their websites, while Chinese CNVD feed is updated only once every week, misses data fields and does not include updates to previously published vulnerability entries. It is also only available for regstered users, with open registration. The second Chinese database, CNVD, also shares data its feeds only to the registered users but does not provide open registration, so we were unable to explore them. Therefore, we had to use web scraping to get data from both Chinese databases. Lastly, Russian BDU shares all its entries in a single Microsoft Excel spreadsheet. However, after examining their data we decided that they are of lower quality and present some issues when trying to incorporate them into our database. For example, they clustered large numbers of CVE references into one entry, which collides with our assumption of using only one CVE ID per vulnerability entry. Moreover, unique IoT related entries were very scarce in BDU, so we decided to abandon it. During our database development NVD launched API to access their data, but for our purposes it provided no benefits over the data feeds.

The remaining, smaller vulnerability information sources, as well as every exploit source, have to be harvested through web scraping. Since their structures vary greatly, we had to develop independent Python scripts for each of them. In most cases data is received using Python requests library, with the exception of Chinese CNVD database, which requires JavaScript code execution to access the data – we use a web browser automated with Selenium for this task. The data received form each source is stored in Raw databases. They are unique for each source and their data structure is intended to match their respective sources. The original data is not modified when entering Raw databases, with the exception for non-English sources, in which case thay are translated before being stored in a database.

Since these Raw databases are incompatible with each other, we introduced an intermediate step before aggregating them. It is data standardization, in which we transform Raw entries into a common Low database format. Some data changes are introduced at this point, for example references are parsed to create list of external IDs, affected products lists are parsed to fit a common format and expanded using products found in titles or descriptions. Some Raw entries that do not contain any particular vulnerability or exploit are omitted. Entries which originally referenced more than one CVE ID are divided into multiple Low database entry, each including only a single CVE.

### 2.1.2 Building Medium and High database

Medium database entries contain information combined from Low databases. To match information, our algorithm compares external identifiers in the Low databases entries. If two or more entries share at least one external ID we assume that they are referring to the same vulnerability or exploit and combine them. However, the algorithm restrains matching entries that share

some external IDs but have different CVE IDs. In such situation, a separate Medium database entry is created for every given CVE ID.

The first algorithm for creating Medium database was iterating over entries in the Low databases one by one. For each Low database entry it was checked if it is already included in any Medium database entry. If not, a new Medium entry had been created. If yes, we had to check if it had to be just updated, merged with other Medium entries (because new external ID emerged in the Low entry) or removed from any Medium entries (because some external IDs have been removed or different CVE ID has been added to the Low entry). Unfortunately, this method was not scalable and in the case of too many matching Low entries it was hard to truncate the resulting Medium entry to include only information from the most valuable and trustful Low entries, so it could fit into the database.

Current algorithm is based on graphs. The graph is built with networkx library, using information about external identifiers. Graph nodes are Low and Medium databases entries plus artificial nodes created from external identifiers found in Low and Medium databases entries. Edges of the graph are defined by relations between Low and Medium databases entries and related external IDs. Nodes and edges have additional attributes (based on the information from the entries) needed to split the graph into Medium database entries.

To get Medium database entries the algorithm selects disconnected subgraphs. If such subgraph has no CVE ID or contains only one in the attributes of nodes and edges, it can be turned into a Medium database entry. If the subgraph contains multiple CVE IDs it is split by edges and nodes with a given CVE ID.

Building graph is parallelized, but splitting and updating the Medium database is sequential. This trade-off is due to keep database entries' identifiers in order, as some new have to be created while others are merged together or deleted. It would be difficult to solve all race conditions if parallelized. Another advantage of the graph approach is a better control over the size of resulting entries. We know upfront which Low databases entries make up a Medium database entry so we can truncate it if needed before uploading to the database.

The High database is updated after the Medium one is ready. This allows us to fully parallelize the process of building the High database thus making it faster.

### 2.1.3 Datasets maintenance automation

Updates of all databases (Raw, Low, Medium and High) are done with Python scripts running automatically according to the schedule. Every Python script updating some database has its own systemd service and related timer triggering its execution. Using systemd services instead of cron gives us better control over which script is currently running, e.g. script building Medium database will not run in its scheduled time if some related Low database updating script is still running – and the same the other way. It ensures that Low databases are up to date and Medium and High databases are built on the stable data.

Process of updating databases is monitored with Metricbeat, which collects logs from the server and uploads them to the Elasticsearch instance where logs can be filtered and presented on the Kibana dashboard. Thanks to the ElastAlert 2 alerts are sent automatically when errors appear in the logs. Alerts are sent to the Slack channel where the team of developers can see them and mark them as "work in progress" or "handled". Another kinds of alerts are these related to the number of entries deleted from the databases and entries disturbingly increasing their size. These errors may be due to some changes in the source data format or bugs in the

scripts. As these problems can be spotted after the database is updated, we need a backup to which we can revert in case we could not fix the erroneous changes. Backup of databases is done with Elasticsearch Index Lifecycle Management. It can automatically create database backups according to the schedule, as well as delete the old ones (the criteria may be when the backup has been created or how many backups already exist). Database backup can also be triggered from the building script.

These automation, monitoring, safety measurements and keeping data in the self-replicating Elasticsearch cluster make whole system resilient and ensure its high availability.

## 2.2 Scan the Internet for IoT devices

Shadowserver performed a variety of tasks in the VARIoT project associated with large-scale IoT device discovery and collection of information on infected devices and IoT malware. It is worth noting that these activities were carried out on an Internet-wide scale, with very large amounts of information captured and shared on a daily basis.

### 2.2.1 IPv6 scanning

IPv6 scanning was for many years considered impossible. And that is correct – blindly scanning IPv6 space is of course, unfeasible. Total IPv6 space is $3.48 \times 10^{38}$ unique addresses. With our current capabilities, it would take roughly $2 \times 10^{25}$ years to scan the entire IPv6 space. Compared that to scanning all of IPv4 space (only about 4.3 billion, out of which we scan 3.7 billion addresses), which nowadays typically takes us minutes!

We chose an approach based on IPv6 hit-list scanning, which has worked surprisingly well. We chose to conduct our scanning based on hitlists of IPv6 addresses observed being used. We compile such a list of /128 addresses from various internal and external sources, which include:

- DNS AAAA records (passive DNS)
- Certificate transparency streams
- The IPv6 hitlist (see https://ipv6hitlist.github.io)
- Sinkholes
- Other lists sourced from partners

We quickly discovered quite a lot of immaturity in the IPv6 scanning tools available. The code quality and amount of tools is lacking compared to IPv4 counterparts. We also found that it is typically much slower to scan for IPv6 than IPv4 – typically over 4 times slower.

Our selected scanning method is biased, due to the fact that we are scanning a) only active IPv6 b) often addresses that have DNS records associated with them.

Hence we are likely missing a large potential set of devices. New methods will need to be introduced to uncover those.

### 2.2.2 Device identification (fingerprinting)

With the unique challenge of scanning every accessible device on the planet and making an assessment as to its make-and-model, we had to find a scalable and robust solution to the problem. It quickly turned out to be unfeasible to use features like nmap OS detection, which would be much too slow on an Internet scale.

After some experimentation we came up with the idea that we would use the corpus of data already being collected in our scans. We started to match various fields/values collected in the scans and often found that these could be used to identify devices at a satisfactory level. A data driven approach with searching for patterns turned out to be the best approach. A human element was necessary to visually verify devices by connecting to them over Tor.

We quickly discovered that enhancements to existing HTTP scans were necessary to be better able to identify certain devices. This especially required our HTTP scanners to be modified to follow redirects and collect favicons.

A lesson learned was that port forwarding was quite common. This meant that different scans on different ports/services, even just HTTP scans themselves, would uncover different devices responding on a single IP.

Another lesson learned is the surprisingly large number of various IP cameras, NVR/DVRs, CCTV that are accessible – the second device category after home routers.

## 2.3 Capture IoT traffic

IoT devices come in various shapes and sizes, the latter meaning that their computing and storage capabilities may also vary from one device to another. And so does their connectivity. Additionally, devices may implement a single function or may provide several functions. And these functions have, as well, various communication patterns, that are dependent on both their connectivity and functions. These three aspects are mainly to be considered to dimension a collection facility for IoT devices.

### 2.3.1 Infrastructure

**Communication Technologies.** Several different communication technologies are commonly used by IoT devices and be usually classified using 2 of 3 aspects: low-energy consumption, high data rate and wide area coverage):

- **traditional cellular devices** use common mobile communication technologies which enable to transmit data at a high rate on wide area. These communications usually include 2G, 3G, 4G, 5G networks and beyond;
- **short range devices** communicate using low-energy and high-rate technologies. The devices can only communicate to a limited area of several meters using Wi-Fi, Zigbee or Blueetooth;
- **LPWA devices** strive to communicate as efficiently as possible and as far as possible, only transmitting a small amout of data per communication period. Technologies include Sigfox and LoRa, but also NB-IoT and LTE-M.

For many consumer IoT devices, communication technology options are quite limited: Wi-Fi, Zigbee or Bluetooth (or Bluetooth Low Energy aka *BLE*) are usually available. But in terms of traffic captures, the cheap and easy approach is to collect traffic at the gateway (or access point). This is usually how other IoT traffic collection initiatives work (see IoT Inspector[1] or TON_IoT[2]). It enables the capture of raw TCP/IP traffic from the initial setup of the device.

In that perspective, we have set up a small computer, a Raspberry Pi[3], receiving traffic mirrored by the wireless access point to which the IoT devices are connected. This enables the capture of TCP/IP communications between the IoT devices and the Internet, and all internal communications transported over Wi-Fi.

**Throughput and Storage.** This modest setup is quite sufficient to collect TCP/IP traffic from 15 to 20 IoT devices (ranging from devices as simple and quiet as smart plugs or motion sensors, to noisy and interactive devices such as voice assistants) for several days. The wireless access point itself supports signal rates ranging from 11 Mbps to 450 Mbps, and its WAN port supports traffic throughput from 10 to 100 Mbps. We have not witnessed any loss in communications as it is highly unlikely that all devices communicate at the same time, be it in their idle state (e.g., all devices receive updates simultaneously) or in their active state (all users employ devices synchronously in a way that it prompts communications towards the access point, or beyond, towards the Internet). All traffic is being mirrorred to the collection device (a Raspeberry Pi) through a LAN port supporting 10- to 100-Mbps throughput.

Depending on the intensity of the interactions with the devices, and the type of devices being sollicited, the amount of captured traffic varies from tens of Megabytes to half a Gigabyte. Of course, the more the voice assistants are used, the more traffic is generated towards the Internet, and subsequently captured. We have actually, because of the COVID-19 imposed lockdown, experienced several deployment environments and levels of interactions : from a single-user, single-room to a frequently visited office room of 3, to a less frequently break room hosting 1 to 5 people, to a 3-people family in a 3-room apartment. Lockdown periods have shown decrease in use and traffic generation when the IoT devices and collection system were deployed in the office premises. Conversely, lockdown periods have seen intense usage in home deployments.

In order to prevent memory depletion of the collection device, the packet captures were migrated daily to a secure storage cloud server. This server is also hosting the data sharing platform[4] for researchers who wish to use our data. The data is available at the granularity of the daily packet captures. The synchronization to the remote server was an intense operation and often delayed the start of the next daily capture by half a dozen minutes at most.

**Collection Period.** As the collection system is quite simple and the interactions with the platform quite frequent, and given the reasonable amount of data produced, we have decided to carry out the collection on 24-hour, 7-days-a-week basis. The IoT platform and its collection system are rarely off. We did have some partial or missing days of collection due to events as various as (un)planned power outages, collection system failure (hardware or software issues), planned maintenance, etc. We did set up a redundant collection system as we considered that the amount of generated data was satisfactory, that we could allow for some losses.

[1] https://inspector.engineering.nyu.edu/

[2] https://research.unsw.edu.au/projects/toniot-datasets

[3] We used the model 3 A+, see https://www.raspberrypi.org/app/uploads/2018/11/Raspberry_Pi_3A_product_brief.pdf for further details.

[4] The VARIoT data portal at Télécom SudParis, IMT: https://variot.telecom-sudparis.eu/.

As per the VARIoT contract, the collection and sharing of our data will end when the project ends. We are working to keep producing data internally and considering avenues to extend data sharing. Additionally, we are also interested in improving the quality of the datasets produced.

## 2.4 Analyze IoT malware

### 2.4.1 Identifying IoT attacks and infected devices through honeypots

Many IoT threats propagate by attacking exposed devices. This made our honeypot network ideal for observing IoT related attacks.

We quickly learned existing honeypots were insufficient to capture these attacks, especially as many of these attacks are focused on various types of Web management interfaces exposed by these devices – routers, cameras, NAS, smart home management systems, PLCs, etc. These management interfaces could be found on many random ports. This encouraged us to introduce a new proprietary honeypot called *Agnus*, which was able to listen to all ports and automatically recognize the protocol being used in a query. A specific protocol handler would then be invoked. Queries could then be tagged by an analyst that would identify a potential attack, exploit and vulnerability.

We started to add various CVE tags to the data being processed and uncovered a wealth of attacks flowing non-stop against IoT and other devices. These included many of the latest exploits but also surprisingly very old ones, sometimes even from the the 1990s. One lesson of note however is that it is becoming increasingly necessary to develop personalities of specific devices, in order to fool an attacker who builds hit lists of devices by scanning, before actual exploitation. Otherwise some of the new attacks will not be observed. This was noticeable in the use of one of the popular open source SSH/telnet honeypots (*Cowrie*) which was more effective at IoT attack observation when different features were randomized to distinguish instances from the *vanilla* version.

### 2.4.2 Capturing IoT malware

Capturing IoT malware at scale is not an easy task. The initial challenge involves extracting a malware callback address (typically an URL) that downloads a script that typically launches subsequent callbacks from various sites that attempt to execute malware binaries compiled for different architectures (x86, MIPS, ARM, etc.). The initial callback resource is typically embedded in the initial exploit. It is not a trivial task to automate this process.

We also ended up building an entire overlay system for malware downloads over our Agnus and Cowrie honeypots. This is an event-driven system decoupled from the honeypot code. The system downloads malware via Tor and monitors the malware C2/malware download endpoints.

### 2.4.3 IoT DGA malware sinkholing

When writing the proposal, we envisioned DGAs becoming more popular in IoT malware. This turned out to be only partially true, as IoT malware has focused primarily on using IPs for C2 purposes. Nevertheless, a few IoT related DGA malware threats did appear. We sample

sinkholed a major IoT DGA threat (*Flubot*), and shared data with the community. We also sample sinkholed *Sharkbot*, and also shared data on *QSnatch*, sinkholed by partners.

### 2.4.4 IoT Malware Analysis

As for the system to analyse IoT malware, we have extended a Linux-based sandbox system in order to support 6 different device architectures that IoT devices use (MIPS, ARM, i386, x86, AArch64, PowerPC, M68K, SPARC and SH4). Each malware received from the Shadowserver Honeypots is run during 2 minutes and all features related to the static binary information, dynamic interacting with operative systems, and all network related information is extracted. We are able to visualize and model the malware campaigns, including the external agents (download servers, c&c servers, dns, other connected services) relationship with each binary and with each other.

We have learned that the actual IoT related malware behaves in a quite simple way, and it does not need operating system specifities (just to be run on a given architecture) nor device specific attacks have been seen (targeted attacks). We have seen that most of the attacks use the IoT devices as an instrument to spread the malware itself, and provide reflection to other types of attacks.

On one side, it would have been interesting to obtain other type of IoT malware from other sources (VirusTotal for instance), and not only from the Honeypots leveraged by Shadowserver, as they are intentionally offering telnet, ssh, and some known vulnerabilities to the attackers, while we could not experiment with other type of malware, like zero-days, or targeted binaries trying to use IoT specific peripherals. Nevertheless, peripheral modeling in embedded device emulation keeps being an intense research topic.

## 2.5 Publish and share IoT data

One of the milestones of the VARIoT project was to publish data on the European Data Portal (EDP; https://data.europa.eu/). European open data initiative aims at making public institutions share as much data as possible so third parties could facilitate them. Data can be statistics, budgets, indexes of objects, etc. Data should be as open as possible, which regards the data format and license. Data format should be machine readable, e.g. CSV, JSON, XML or, even better, JSON-LD or RDF, which are enriched with context helping to understand the data found in the file. License should also ease the use of data, so it should be as permissive as is possible, e.g. one of the Creative Commons licenses. Every dataset should be provided with metadata. This metadata should consist of: title, description, tags, license, author, dataset creation and update timestamp. It is also a good idea to translate the metadata into as many languages as possible for better accessibility of the dataset worldwide.

The easiest way to publish own data is to get in touch with your national open data portal (NDP). All datasets published there are automatically imported into the EDP catalogue. In case of a large number of owned datasets or a need of high automation with publishing them, one can decide to host own open data portal which will be crawled by the European or a national one. Both ways have their pros and cons. Using national data portal is as easy as having an account and maintaining entries with own datasets. Datasets owner is only responsible to keep data accessible and provide meaningful descriptions in the national data portal. On the other hand, one does not have control over how dataset is presented on the national and the European data portal, as it depends on their CMS and how well they are compatible with each

other. To have more control over datasets presentation and integration with EDP one can host their own data portal. However, it requires one to maintain and secure this installation and this adds some extra work.

To share data produced in the VARIoT project we have decided to use national data portals. It is easier for each partner to cooperate with the data portal from their own country, as well as it is in line with the spirit of the whole open data initiative to make more public institutions share data on their national data portals.

Joining national data portals and sharing data there was relatively easy. We had more troubles with providing high quality metadata on the EDP side because of many problems with integration between EDP and NDPs. Even though every data portal should be compatible with DCAT-AP metadata standard (https://joinup.ec.europa.eu/collection/semantic-interoperability-community-semic/solution/dcat-application-profile-data-portals-europe/release/210) it turned out that some metadata present in the NDPs was absent from the EDP. We had to ask people who maintain data portals for help to check or fix integration of their portals with EDP. For example, most of the problems with integration between Polish data portal (PDP) and the EDP were fixed after EDP started downloading data from the new PDP API. However, some information present in the PDP is still missing from the EDP, including the most important ones, i.e. the shared files type and the dataset license details.

# 3 IoT Data Processing

Within VARIoT, numerous datasets were produced reaching several large volumes leading to data processing and storage challenges. This section describes how these challenges were addressed in VARIoT.

## 3.1 Big Data processing

Frequently created datasets in VARIoT are pcap files and CSV files. Commonly used tools for processing these files are `tshark`[5], `argus`[6], `zeek`[7]. In case of a large volume of files, it is time consuming to process them one by one sequentially.

The `tshark` tool could be well executed in parallel thanks to the `parallel`[8] tool. In some cases, it is time consuming to extract all features of all packets, if only on the packets sent by the IoT devices and not the other devices that are concurrently sending messages during the packet capture. In that case, the packet captures can be preprocessed with `tcpdump` to have a reduced data set for feature extraction.

An example is shown below

```
find pcaps/ -type f | sort | parallel -j7 extract.sh {}

#extract.sh
T=‘echo $F | sed ’s#/sensors/#/anlysis/pcaps/#g’ | sed ’s/.gz//g’‘
D=‘dirname $T‘
mkdir -p $D
zcat $F | tcpdump -n -r - -w $T "‘cat␣filter‘"
```

In this example, a list is done and sorted. Blocks of 7 pcaps are taken and 7 instances of the script `extract.sh` are executed. The pcaps are decompressed and `tcpdump` is executed on each pcap file filtering out the relevant traffic defined in the file called `filter`.

In the example, the features timestamp (time_epoch) and source IP are extracted from a pcap file in parallel.

```
find analysis/ -type f | sort | parallel -j 7 parse.sh {}
#parse.sh
T=‘echo $F | sed ’s#/source#/parsed/#g’ | sed ’s/cap$/txt/g’‘
D=‘dirname $T‘
mkdir -p $D
tshark -n -E separator=’|’ -r $F -T fields -e frame.time_epoch -e ip.src > $T
```

[5] https://www.wireshark.org/docs/man-pages/tshark.html
[6] https://openargus.org/
[7] https://zeek.org/
[8] https://www.gnu.org/software/parallel/

**Distributed counting.** A commonly used activity performed with VARIoT datasets is counting various features. When the counting is done in parallel, a special care must be taken to avoid incorrect counting. It should be recorded which files were already processed to avoid that the counter is increased when the counting distributed process is started a second time. It is recommended to use systems capable of handling concurrent accesses for the counting. An example is the usage of a tool such as `redis`[9]. An example is shown below. In this case a list of features extracted from packet captures are processed. Each feature file is processed with the script `record.py`. It uses the day of the timestamp as key and counts for this day how many times an IP address was seen. The concurrent access is handled by the tool `redis`.

```
find parsed/ -type f | sort | parallel -j7 record.py {}
```

```
for line in open(sys.argv[1], "rb"):
    (epoch, ipsrc,ipdst) = line.split(b"|")
    t = datetime.datetime.fromtimestamp(float(epoch))
    day = bytes(t.strftime("%Y%m%d"),"ascii")
    red.zincrby(k,ip.src,1)
```

**Parallel processing pitfalls.**

**I/O (Input / Output)** When large data chunks are read from one disk in parallel and each job writes on the same disk, the I/O limits are quickly reached. In case I/O limit is reached, jobs spend most of their time waiting for I/O, rendering parallel processing ineffective. A best practise is to monitor I/O such as with the tool `iotop`.

**Re-usage of partial results** Sort the input lists and store them. In case of failure of distributed processing, you could still reuse the results of the previous runs.

**Accidental overwrite** In some cases, one output could overwrite the results of another process. Hence, use different outputs and merge or synchronize them.

**Buffering** The tool `parallel` can buffer the output of a job output to avoid a mix with other jobs. In case the job is more verbose, more results have to be buffered. Hence, the output of each job could be written in a dedicated file to avoid output mixing and memory exhaustion.

**Race conditions** Avoid race conditions, especially in distributed counting where two processes update the same counter at the same time.

## 3.2 Storage

In VARIoT continuous network packet captures were done over long periods of time.

### 3.2.1 Network packet captures

In this section is written how network packet captures were done. There is a magnitude of tools for doing network pcap captures in *nix operating systems such as `tcpdump`, `argus`, `zeek`,

[9] https://redis.io/

`tshark` and `wireshark`. However, many tools rely on a few libraries such as libpcap[10] or wiretap[11].

While doing network packet capture, it is essential to document on which network interface the network packet captures were done in a network diagram or similar documentation to have an overview of the completeness of the packet captures. For instance, if a packet capture is done on a network interface behind a packet filtering device, the network capture might be **incomplete** in case the packet filtering device removed some packets. In addition, the `snaplen` parameter is important to define how many bytes are captured. If the `snaplen` is smaller than the actual packet, then the packet is truncated. In some cases, when capturing high volume packets, the `snaplen` is put smaller but the content of the packets are lost. In other cases, the complete packet is desired. In case the feature packet length is used in machine learning, it is essential to verify the `snaplen` to avoid corrupted models.

The snaplen parameter could be recovered with a detailed analysis of the packet captures by comparing the libpcap header values, snaplen and len[12]. Those record header values are prepended to the data of the packet captures. The record header has also **timestamps** when the packet capture was done. If the packets are reprocessed with other tools such as TCP reassembly tools or intrusion detection tools, timestamps are added to the generated data. However, these timestamps reflect the time when the tool was executed and not when the packet arrived on the wire. Hence, it is recommended to compare the timestamps to ensure that they are not modified during the processing.

Space issues of very large files are frequently addressed with *file rotation* features of the packet capturing tool. The tool `tcpdump`[13] supports the rotation either when a given file size is reached with the (-C command line switch) or after a given elapsed time (-G command line switch). Regardless of the option used, there is a risk that a directory quickly runs full of files. To address this, packet capturing tools have often a post rotate command that can be executed. The tool tcpump has the (-z command line switch). It is recommended to use this switch with a tool of minimalistic logic to avoid jeopardizing the packet capture in case of crashes of the post execute command tool or if there are memory leaks.

Long lasting packet captures can be done with `tcpdump` streaming to standard output with the command line option (-w -). Then pipe the output to d4 client [14] [15]. The tool `d4` client is capable of doing continuous packet captures over several years without interruptions. When relying on network captures in real time applications, the streaming to standard output of `tcpdump` might hit some buffering issues. The packet is in a buffer, not streamed yet as the buffer is not full. Those issues can be addressed with the (-U command line switch).

An example of `tcpdump` streaming is shown below. A special attention must be taken to exclude the traffic that is sent to the collector.

```
tcpdump -l -s 65535 -n -i vr0 -w - '(␣not␣port␣$PORT␣and␣not␣host␣$HOST␣)' |
   socat - OPENSSL-CONNECT:$COLLECTOR:$PORT,cert=/etc/openssl/client.pem,cafile
   =/etc/openssl/ca.crt,verify=1
```

[10] https://www.tcpdump.org/
[11] https://gitlab.com/wireshark/wireshark/-/tree/master/wiretap
[12] https://www.tcpdump.org/manpages/pcap.3pcap.html
[13] https://www.tcpdump.org/
[14] https://github.com/D4-project/d4-goclient
[15] https://github.com/D4-project/d4-core/tree/master/client

The file rotation of pcap files might lead into a case of disruption of stateful protocols such as TCP. Hence, it could be that in the first file was the establishment of the TCP session and in the second file the session itself. When those files were processed, the TCP session looks empty in the first file, in the second file it might be discarded as the session establishment is missing. To address these issues, some pcap processing tools could either process the `tcpdump` stream directly or multiple files. The tool `pcapdj`[16] takes a list of multiple pcap files as input and streams them in a named pipe. Hence, the tool doing stateful analysis does not see if the packet capture is split over multiple files.

In VARIoT, despite the fact that CSV with machine learning features are provided, the original format is also preserved and published. Hence, researchers could even extract other features that were uncommon or unknown. Hence, a best practise is to keep the data in the most original form as possible.

The packet captures best practises are summarized below.

**Completeness** Watch out for the snap length and the interface where the packet capture is done.

**Feature verification** Check the network captures if the extracted machine learning features are not biased by artefacts in packet capture such as execution timestamps instead of original timestamps, truncated packets and so on.

**Timezones** Take into account time zones of the machine where the capture was made when doing temporal correlation with other timestamps.

**File rotation** Use a file rotation to avoid file system issues.

**Post execute** Implement the logic to handle the recently created files such as pushing the recently created filename in a queue.

**Stateful protocol analysis** Check if your packet analysis include the analysis of stateful protocols where states might spread over multiple files.

**Original data format** Keep the data in the most original form as possible to avoid data loss.

### 3.2.2 PCAP storage

pcap files easily grow and consume a lot space on disks. Either they can be truncated with the usage of a smaller snaplen option or they could be stored over multiple machines. A minio storage was setup in VARIoT to store the pcap files. Minio is a Multi-Cloud Object Storage[17]. In D4, the concept of analyser exists [18]. A minio analyser was configured. Each time a file was written on disk, the D4 server updates the analyser queue that was configured. Each analyser has a UUID and a type. An external tool can then derive the queue name and poll the queue. The advantage of `minio` is that the pcaps can be distributed and replicated over multiple nodes. The buckets should be dimensioned accordingly to the expected incoming volume. Small packet captures can be stored on one replicated cluster for an entire month. Other captures include packet captures of one day.

An example is shown below to push pcap files stored by d4 to a minio cluster.

[16] https://github.com/CIRCL/pcapdj
[17] https://min.io/
[18] https://github.com/D4-project/architecture

Minio is compatible with Amazon S3 storage[19]. Hence the pcaps could easily be stored with the same cluster in an Amazon cloud instead of a local one. The tool `mc`[20] of minio enables a command line interface to copy pcap files from a minio cloud to a local file system. An example is shown below on how to store pcap files in minio.

```
client = Minio(config.minio_url, access_key=config.minio_access_key,
              secret_key=config.minio_secret_key,secure=False)


red = redis.Redis(host=config.d4_redis_srv, port=config.d4_redis_port, db=config
    .d4_redis_db)

#Iterate over queue and push all files to minio
while True:
    obj=red.rpop(config.d4_redis_queue)
    if obj is None:
        break
    targetfile=codecs.decode(obj,'utf-8')
    if targetfile is None or targetfile == "" or targetfile == "None":
        break
    target=targetfile.replace("/disk1/","")
    syslog.syslog("PUSHING␣to␣minio:␣" + target)
    client.fput_object(config.minio_bucket, target, targetfile)
    time.sleep(1)
```

## 3.3 Privacy

In VARIoT, the focus was set on the collection of data on IoT devices. Network packet captures were made. Signatures were generated and exposed and exposed devices were identified. From the network captures, features were extracted. Most network captures were done in lab environments excluding personal data. The recorded IP addresses are by majority RFC1918[21] private IP addresses. The public IP addresses are the IP addresses of legal entities distributing firmware updates. The scanned IP addresses have as meta data country data and do not focus on the identification of the person operating these IoTs. The exposed and infected devices were shared with CSIRTs having a legitimate interest in terms of GDPR to process such data. The purpose is to identify stakeholders that are capable of fixing the security issues. The data can also be shared with MISP supporting the sharing of personal data [22].

In the case of IoT data collection involving human subjects, we had to deal with several points of attention, in particular with respect to GDPR. Indeed, whenever you are involving the public (even restricted to internal staff members) into an experiment, you should contact your Data Protection Officer at your institution to discuss the settings of the experiments and its impact on privacy. A formal privacy impact analysis (*PIA*) could be conducted to identify potential privacy

[19] https://aws.amazon.com/s3/
[20] https://github.com/minio/mc
[21] https://datatracker.ietf.org/doc/html/rfc1918
[22] https://www.misp-project.org/compliance/GDPR/

infringements. Even when no personal information is recorded, the public must be informed that data is collected based on their actions. Typically, our lab environments were set up in generic office with department-restricted access. All department members were asked to fill in a consent form before accessing the room and using the deployed IoT devices. As mentioned, interacting with the IoT devices would generate traffic within the internal network or towards the Internet. The latter traffic was captured. We ensured that the outgoing traffic was not containing any sensitive information, and if it was, that the data was encrypted. Discussing the matter with the DPO, we revised our position on using voice assistants which were sending voice commands towards extra-European servers to be processed there. Although, the related packets were encrypted, they are eventually decrypted on the server side for the voice command to be processed. The DPO took a conservative measure as there was no evidence that the voice data, which can be tied to a person, and therefore constitutes personally identifiable information (PII), was erased once processed. The consent form was written to inform people on what and how the data was collected, and for what purpose. It also mentioned the date of the end of the project as the end date of the collection.

Upon sharing the data with our national data portal and the European Data Portal, we met again with the DPO to validate that the data to be shared was not containing anything personal. In order to ensure better control, IMT shared CSV data that are representative features modeling the legitimate traffic of IoT devices with the data portals. Packet captures were kept on our own data portal and requires a motivated registration from identified researchers. The registration is reviewed and processed manually, with credentials exchanged over an encrypted channel.

# 4 Conclusion

Data is an invaluable resource that needs specific attention to be actionable, especially in the cybersecurity domain where it gives a particular edge against cyber attackers. In VARIoT, we have developed the means to collect, process, fuse, correlate, store and share them in a large-scale, efficient and privacy-preserving way, when needed. In the process, we have improved our approaches to comply with the requirements of the project and to contribute to the accumulation and discovery of new knowledge, and the raising of awareness on the vulnerabilities of IoT devices.

In this report, we have gathered feedback from our experiences and tried to produce new knowledge that will be useful for practicioners that seek to reproduce our work, be it in the cybersecurity domain, or in the realm of data collection and sharing in general.

We have demonstrated how it is possible to collect heterogeneous data and correlate them in order to extract new knowledge. We have explained how vulnerabilities of IoT devices can efficiently discovered using scanning. We have gone over the techniques and pitfalls of capturing IoT in both closed and realistic environments and how to analyze relevant IoT malware in a safe way. Finally, we have shared our experience using information sharing platforms for publishing IoT data. We have completed this feedback with more general considerations in data processing that should concern practitioners that would encounter issues as diverse large amounts of data process or store, and personally identifiable information.


## Consortium

NASK

SHADOWSERVER

CIRCL
Computer Incident
Response Center
Luxembourg

Mondragon
Unibertsitatea

TELECOM
SudParis

IP PARIS

## Disclaimer

All information provided reflects the status of the VARIoT project at the time of writing and may be subject to change.

Neither the VARIoT Consortium as a whole, nor any single party within the VARIoT Consortium warrant that the information contained in this document is capable of use, nor that the use of such information is free from risk. Neither the VARIoT Consortium as a whole, nor any single party within the VARIoT Consortium accepts any liability for loss or damage suffered by any person using the information.

This document does not represent the opinion of the European Community, and the European Community is not responsible for any use that might be made of its content.

## Copyright Notice




**Co-financed by the Connecting Europe Facility of the European Union**