SOCCRATES

SOC & CSIRT Response to Attacks & Threats based on attack defense graphs Evaluation Systems

D4.2 Threat Identification and Threat Trend Prediction – Initial Prototype

Deliverable type:	Demonstrator
Contributing work packages:	WP4
Due date of deliverable:	31/08/2020
Submission date:	31/08/2020
Dissemination level:	PU

Responsible organisation:	SHS
Editors:	Piotr Kijewski
Revision:	1.0

Abstract	This deliverable provides an overview of the initial threat identification and trend prediction toolset prototyping carried out as part of Task 4.2 in WP4: Threat Identification and Threat Trend Prediction. A set of Dockerized tools utilizing ML and string similarity algorithms has been developed for the purpose of identifying and classifying DGA-based domains in large domain datasets derived from The Shadowserver Foundation's malware sandbox.
Keywords:	Machine learning, malware, DGA, sandbox, botnet

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 833481
Call H2020-SU-ICT-2018 · Innovation Action • Start date: September 1st, 2019

Management summary

This deliverable provides an overview of the initial prototyping work carried out as part of Task 4.2 in WP4: Threat Identification and Threat Trend Prediction. Shadowserver collects around 500 000 (presumed) malicious samples per day as part of its mission of fighting Internet threats. The total current repository size (as of the time of writing this deliverable) is 1.3 billion samples. The samples are executed dynamically in a sandbox environment, with a report being created for each sample that includes observed networking behaviour and host system changes. This includes the monitoring of domain names that samples attempt to connect to. Some of these domains are Domain Generation Algorithm (DGA) domains, used by cybercriminals to improve resilience of botnet (malware) Command and Control (C2) networks.

The challenge is to be able to perform industry scale classification of these domains - specifically whether these domains are generated using DGAs and if so, which malware family is behind their usage. Identification of a previously unknown DGA and subsequent reverse engineering of malware that utilizes these domains provides valuable threat intelligence for identifying infected computers.

To facilitate the above a Machine Learning (ML) Long Short Term Memory (LSTM) and string grouping based initial prototype toolset has been developed to identify DGA domains and classify DGA domains by malware family from a list of domains observed. Preliminary results show a 96,5% precision in identifying DGA domains with a more varied result for classifying specific families. As it was recognized that a single mechanism for detection and classification is not a viable approach to the problem, a DGA processing pipeline has been proposed in the form of a Dockerized framework that can be used for easy deployment and testing of different detectors/algorithms.

Once the prototypes are deemed satisfactory for large scale production usage, they will be run regularly over Shadowserver datasets, with results stored at Shadowserver (one of the pilot environments) and made available to the SOCCRATES platform via a to be developed API.

Contributor(s)	Irina Chiscop (TNO) Federico Falconieri (TNO) Piotr Kijewski (Shadowserver) David Watson (Shadowserver)

Reviewer	Martin Eian (MNO)

Security Assessment	**See deliverables table for security assessment requirements**
Approval Date	27/08/2020
Remarks	none

TABLE OF CONTENTS

1 INTRODUCTION 5

1.1 The SOCCRATES project 5
1.2 This deliverable 6
1.2.1 Motivation (Purpose) 6
1.2.2 Scope 6
1.3 Structure of this deliverable 7

2 DGA BACKGROUND 8

3 DATASET 11

3.1 Acquisition and Context 11
3.2 Dataset Availability 12
3.3 DGA Dataset Description 12
3.3.1 DNS queries made by file samples running in the Shadowserver sandbox 12
3.3.2 Unregistered domains seen in sandbox 13
3.3.3 DGA domains from known malware/seeds 14
3.3.4 SSL certificates collected from remote hosts visited by samples running in the sandbox 15
3.3.5 Known top domains 18
3.3.6 Further datasets 18

4 DGA CLASSIFICATION METHODS AND PROCESSING PIPELINE 19

4.1 Goals 19
4.2 DGA classification methods 19
4.2.1 String similarity methods 19
4.2.2 Machine Learning methods 20
4.3 Software deliverables 21
4.3.1 String grouper DGA family classifier 21
4.3.2 LSTM DGA binary classifier 21
4.3.3 DGA pipeline 23

5 IMPLEMENTATION AND DEPLOYMENT 24

5.1 DGA pipeline architecture 24
5.2 Implementation 25
5.3 Deployment 25
5.3.1 Current deployment 25
5.3.2 Production deployment 27

6 PRELIMINARY RESULTS 28

6.1 Performance evaluation 28
6.2 String grouper evaluation 28
6.3 LSTM binary classifier evaluation 30
6.4 Future work 30

7 CONCLUSION AND NEXT STEPS 32

8 REFERENCES 33

9 ABBREVIATIONS 34

1 Introduction

This section introduces the SOCCRATES project and defines the goals of this deliverable.

1.1 The SOCCRATES project

SOCCRATES (SOC & CSIRT Response to Attacks & Threats based on attack defence graphs Evaluation Systems) is an EU funded project under the Horizon2020 programme that has the following main challenge:

> ***How can SOC and CSIRT operations effectively improve their capability in detecting and managing response to complex cyber-attacks and emerging threats, in complex and continuously evolving ICT infrastructures while there is a shortage of qualified cybersecurity talent?***

The main objective of SOCCRATES is to develop and implement a security automation and decision support platform ('**the SOCCRATES platform')** that will significantly improve an organisation's capability (usually implemented by a SOC and/or CSIRT) to quickly and effectively detect and respond to new cyber threats and ongoing attacks.

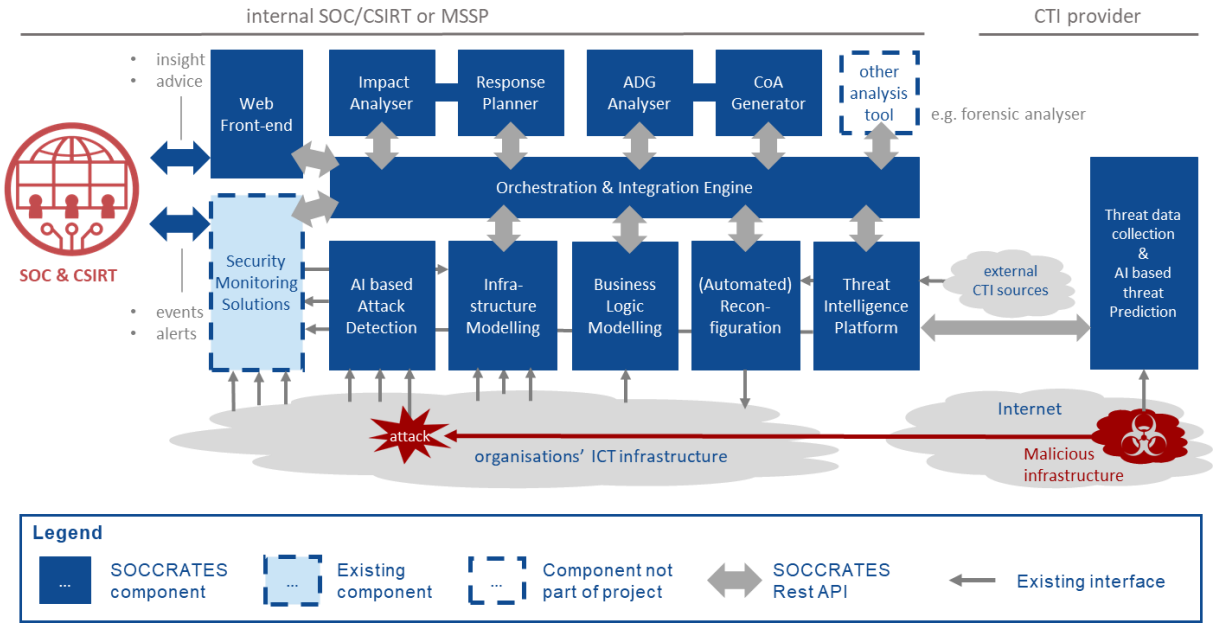

Figure 1.1 – The SOCCRATES platform

The SOCCRATES platform (see **Figure 1.1**) consists of an orchestrating function and a set of innovative components for automated infrastructure modelling, attack detection, cyber threat intelligence utilization, threat trend prediction, and automated analysis using attack defence graphs and business impact modelling to aid human analysis and decision making on response actions, and enable the execution of defensive actions at machine-speed.

SOCCRATES has the following concrete project objectives:

1. Deliver the SOCCRATES platform consisting of an orchestration function and a unique integration of innovative background solutions that seamlessly work together.
2. Show that the SOCCRATES platform can improve SOC operations by evaluating the SOCCRATES platform in two diverse real-life pilot environments.
3. Examine and illustrate the benefits of automation for selected SOC activities to help manage the cyber security skills gap in organizations.
4. Prepare for successful exploitation by the SOCCRATES partners of the individual innovated components and the integrated SOCCRATES platform in commercial products that are offered to the market and are available for the European (business) community.

Please visit www.soccrates.eu for more information on the SOCCRATES project.

1.2 This deliverable

1.2.1 Motivation (Purpose)

This Deliverable is a report on the progress of work done so far in Task T4.2 (Work Package 4) "Threat Identification and Threat Trend Prediction". It is focused on Domain Generation Algorithms (DGAs) used by cybercriminals to operate botnets made up of compromised computers. The goal is to enable industrial scale detection and classification of DGA domains seen in The Shadowserver Foundation's malware Sandbox environment.

This Deliverable document accompanies the initial implementation and deployment of tools to facilitate the above - the actual "Demonstrator" itself. It is intended to serve as a guidance document to the reader. It provides the background of the DGA problem to be solved along with an overview of the proposed algorithms and tools developed so far. A DGA processing pipeline is proposed to chain these tools together as well as an illustration of the deployment process.

Note that while formally the Task was due to start in M6, work on the Task started immediately after project kick-off at M1.

1.2.2 Scope

The Shadowserver Foundation Sandbox stores the results of the processing of around 500 000 (mostly) malicious files per day, for a total of over 1.3 billion samples stored. The meta information collected includes Anti-Virus (AV) classification from a number of AV engines, internal Shadowserver classification (based primarily on YARA rules), Snort/Suricata triggered rules, domains and IPs contacted, as well as host changes observed. Initial data exploration of the Sandbox malware processing from a big data analytics perspective utilizing machine learning techniques is focused on DNS based threats. This means threats that utilize DGA domains for Command and Control (C2) services (either as a primary or backup mechanism) or data exfiltration.

This document covers the results of our initial work into a practical approach to the DGA detection/classification problem in "big-datasets" and our first prototypes deployed. Please note the work described here continues to be a "work-in-progress" and the results are to be considered preliminary.

1.3 Structure of this deliverable

This document is structured as follows:

Section 1 – Introduction – the section you are reading right now.

Section 2 - DGA Background - gives a brief overview of Domain Generation Algorithms, DGA domains and outlines the need to solve the DGA problem.

Section 3 - Dataset - provides more detailed information on the dataset features being explored for the detection and classification of DGA domains.

Section 4 - Algorithms and DGA Processing Pipeline - provides a description of the machine learning and other algorithms (detectors) that have been considered or are being applied and the proposed processing pipeline. In particular, Long Short Term Memory (LSTM) neural networks and string grouping algorithms are explored. A pipeline for processing DGA domains is introduced.

Section 5 - Implementation and Deployment - describes the implementation and deployment of the developed Dockerized[1] toolset.

Section 6 - Preliminary Results - describes the first (preliminary) results obtained.

Section 7 - Conclusions and Next Steps - describes the conclusions and next steps.

Section 8 - References - contains a list of reference material used.

Section 9 - Abbreviations - lists abbreviations used in this document and SOCCRATES.

[1] https://www.docker.com/

2 DGA Background

Domain Generation Algorithms (DGAs) are used by malware authors to programmatically generate many potential domain names, changing the entire dataset every pre-set time period. Knowledge of a DGA, the time period and input seed allows a criminal to determine what unique domain name(s) will be required to be registered on a particular day in order to maintain Command and Control (C2) communications with malware-infected computers. For defenders lacking such knowledge, it is very difficult to know what C2 domains will be used by individual malware samples, until live traffic can be observed (such as after malware detection and collection, in a sandbox). By which point, the criminal will likely already have registered the C2 domain for that day and ensured they have gained or maintained control of their botnet.

DGAs exploit economic asymmetry, meaning that the criminals only have to register one of the programmatically generated possible C2 domains to maintain control of their botnet, since each bot will cycle through all of the possible C2 domains until it does manage to successfully connect to a valid C2 server. For example: the Gameover Zeus [1] banking trojan malware generated 1000 possible C2 domains per week, and the CryptoLocker [2] ransomware generated 1000 possible C2 domains per day.

DGAs pose significant challenges for defenders and Law Enforcement:

To disrupt/break criminal control over all of the malware infected victim devices, the defender must gain control of every one of the potential C2 domains every day - not just the small number of C2 domains actually registered by the criminal. In the Avalanche [3] case, over a 4-year period, over 5.4 million domains had to be seized, blocked from issue or created, in order for German and US Law Enforcement to protect ~5 million users from further exploitation globally. This was, and continues to be, a mammoth undertaking. Failing to register/block even a single domain, on any day, risks that the criminal may potentially regain control of their entire botnet. The potential risk of defensive failures is high.

The potential C2 domains used in DGAs exist within Top Level Domains (TLDs) operated by DNS registries based around the globe, often in countries that are 'unfriendly', where a national law enforcement agency has no authority, or in jurisdictions that simply do not have the legislation to seize or block the issue of domains (the UK is one such example).

The time to take action against malware DGA C2 domains can be very protracted. The Avalanche case took 4 years of investigation before action could be taken. During that time, millions of users were infected daily with one of the 20 malware strains the Avalanche platform was used to deploy and manage. Not only is this time delay bad for the victims, but it is also bad for society - undermining trust in the Internet, trust in online banking and costing the public purse to fund the considerable length of effort required to tackle such an operation globally.

Most of the top global malware-based threats utilise DGAs to make their C2 infrastructure resistant to takedown.

Shadowserver has a malware repository of over 1.3 billion malware samples (unique by hash) and executes hundreds of thousands of unique new malware samples in our virtual and physical sandboxes every day, with live Internet connections and custom DNS failure modes to force domain name usage enumeration. Shadowserver therefore has excellent knowledge and visibility of the C2 domains that running malware attempts to resolve and connect to.

Shadowserver works with industry partners to identify DNS traffic from particular groups of similar malware samples, reverse engineer the DGA code within the malware, re-implement that code, extract the DGA seed value(s) and time period, and then forward predict all the possible future C2 domains for those particular malware samples. We share that data within selected private trust communities. We then remove all of those known potential DGA C2 domains from our sandbox DNS traffic and repeat the process until all observed DGA traffic has been successfully identified, predicted and filtered out. We then wait for new, unknown malware to generate new DNS traffic patterns and restart the process.

Example DGA code (early Cryptolocker):

```
def generate_domain(year, month, day):
    """Generates a domain name for the given date."""
    domain = ""

    for i in range(16):
        year = ((year ^ 8 * year) >> 11) ^ ((year & 0xFFFFFFF0) << 17)
        month = ((month ^ 4 * month) >> 25) ^ 16 * (month & 0xFFFFFFF8)
        day = ((day ^ (day << 13)) >> 19) ^ ((day & 0xFFFFFFFE) << 12)
        domain += chr(((year ^ month ^ day) % 25) + 97)

    return domain
```

Returning:

January 7th, 2014 = intgmxdeadnxuyla

January 8th, 2014 = axwscwsslmiagfah

As of the time of finalizing this Deliverable, Shadowserver and its partners are aware of 91 different DGAs that have been reverse engineered. Together we have extracted and forward predicted 821 different DGA and seed combinations, which equates to over 400,000 different unique potential DGA malware C2 domains per day. At a cost of $10 per domain, it is simply financially impossible to purchase that volume of domains per day (or even sample sinkhole a small percentage of domains per day).

The chart below shows the number of potential DGA domains per malware family per day, which covers the major of top cybercrime threats to world (including many of the most prevalent banking trojans and information stealers):

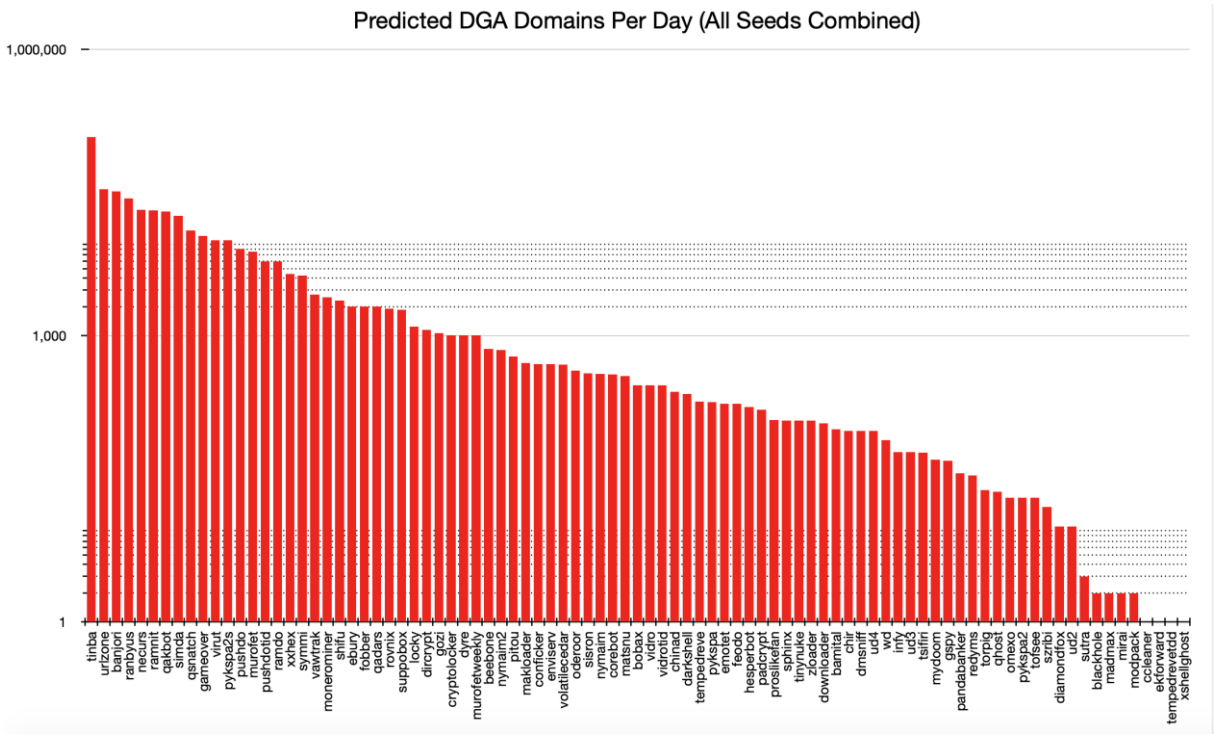

Figure 2. Potential Number of DGA Domains Per Day for known malware families/seeds

3 Dataset

This section describes how the domain and DGA data is collected and made available for analysis. It also provides a description of the datasets made available for SOCCRATES.

3.1 Acquisition and Context

The malware used as a basis for the generated data is collected by Shadowserver primarily through reciprocal bulk data sharing agreements with AV/threat intelligence vendors and researchers/research groups, the Shadowserver Sandbox itself (second-stage and other artefacts retrieved during initial sample execution), honeypots and honeyclients, spampots and submissions from investigations.

At the time of writing this deliverable, the malware repository consists of nearly 1.3 billion unique samples, and the daily average of unique samples ingested over the last 3 months is around 380 000.

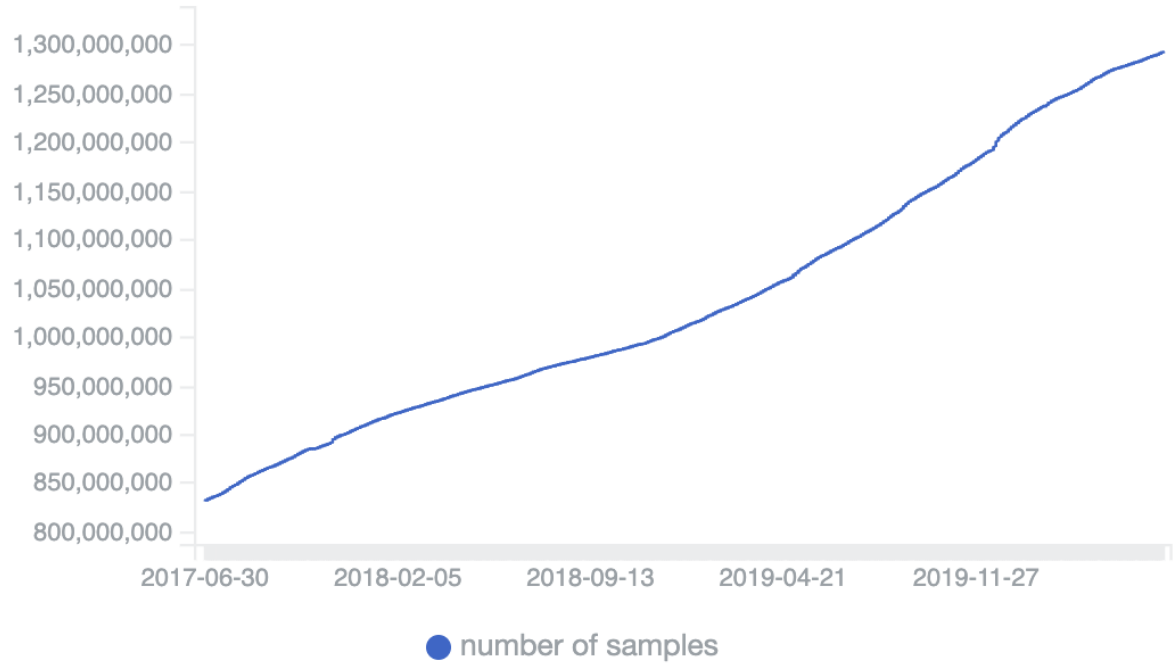

Figure 3. Number of samples stored in the Shadowserver malware repository over time

The dataset being investigated is derived from the results of the execution of malware in the Shadowserver Sandbox environment. No context of how the malware itself was collected is stored.

The main focus of the initial SOCCRATES ML big data investigation is on derived DNS related information, i.e. information about domains that are being queried as a result of the malware runs. This includes potential DGA domain candidates.

3.2 Dataset Availability

For the purpose of the project, a special Virtual Machine (VM) with an ELK[2] setup was made available by Shadowserver for project partners. The VM contains representative datasets to be investigated and used for testing and training ML/AI algorithms. The VM was made available in late November 2019 and a workshop was held in Vienna at that time to explain the datasets to project partners.

3.3 DGA Dataset Description

For the purpose of the Domain Generation Algorithm investigation, the ELK VM with the test/training set data contains the following datasets/sources:

1. DNS queries made by malware running in the Shadowserver sandbox
2. Unregistered domains seen
3. DGAs from known malware/seeds
4. SSL certificates collected from remote hosts visited by samples running in the sandbox
5. Known top domain lists: Alexa, Cisco, Quantcast, etc.

Each of these datasets is described in more detail in the following sections.

3.3.1 DNS queries made by file samples running in the Shadowserver sandbox

3.3.1.1 Description

This index contains DNS queries seen in the Shadowserver Sandbox for the given period. Samples are ingested into the sandbox from various sources and executed in a virtual and/or bare metal setup. DNS queries are captured and indexed, along with associated meta-information about a domain, or connection. Samples are executed in the sandbox for a set period of time. All executions are discrete – i.e., there is no link between individual samples being executed other than overlap in features (which is what we are searching for).

Caveats:

- Not all samples ingested and executed are by definition malware, benign files may sometimes also be executed.
- Many of the DNS queries may be to benign domains, this includes not just those initiated by benign files but also malware itself can make “legitimate” queries or queries to fool analysts.
- AV engines can trigger on samples (the number of engines triggering a rule on a sample is also provided). Note however that this does not mean a sample is malicious, as AV engines can sometimes trigger on benign samples too (thus generating a False Positive (FP) verdict).

[2] https://www.elastic.co/what-is/elk-stack

3.3.1.2 Index fields

Field name	Description
Timestamp	Time of query (UTC+0)
Domain	Domain name being queried
Tld	TLD of domain name being queried
md5, md5hash	MD5 hash of sample performing the query
filetype	File initiating the query
avcount	AV detections that trigger on sample
tag	Additional meta-information on classification
source	Where sample came from
resolv	GEO, IP and ASN of domain if available
sample_timestamp	When sample was executed
sandbox	Type of sandbox (must be Virtual or Metal)
sha1	SHA1 hash of sample
sha256	SHA256 of sample
sha512	SHA512 of sample
timeend	Timestamp when sample run finished

3.3.1.3 Dataset size

Over 11 million records of queries were provided for testing and training.

3.3.2 Unregistered domains seen in sandbox

3.3.2.1 Description

This index contains information on domains that return upon DNS query an 'NXDOMAIN' response. This means the domain does not exist (is unregistered). This does not mean that the domain name is malicious, but it is a strong indicator of maliciousness if such responses are seen in a malware sandbox.

3.3.2.2 Index fields

Field name	Description
md5	MD5 hash of malware issuing the DNS query
domain	Domain being queried
avcount	AV detections that trigger on sample
avtotal	Total number of AVs scanning sample
age	How many days seen requested
samples	Samples issuing query
dga	DGA algorithm identified if any
dga_from	DGA timestamp validity start
dga_to	DGA timestamp validity end

3.3.2.3 Dataset size

Over 12 000 entries were provided for testing/training.

3.3.3 DGA domains from known malware/seeds

There are two sources of DGA domains in the dataset.

1) FKIE DGArchive[3]: these events have the `source` set to `Fraunhofer FKIE DGArchive`. A `rule` should be present.
2) Netlab360[4]: these events have the `source` set to `Netlab360`

3.3.3.1 Description

This index contains information about known DGA domains for the set period. These DGAs have been reverse engineered by analysts. By using the DGA algorithm with the right seed it is possible to predict domains that will be used for a given day. Associated meta-information - such as malware family assignment - is also provided. Some of these domains may also be present in the malware sandbox DNS queries.

[3] https://dgarchive.caad.fkie.fraunhofer.de/welcome/
[4] https://data.netlab.360.com/dga/

3.3.3.2 Index fields

Field name	Description
timestamp	DGA validity start (UTC+0)
domain	DGA Domain
tld	TLD of DGA Domain
tag	May be set to rpz-dga or additional meta information related to malware
source	Where DGA came from
family	DGA Malware family
resolv	GEO, IP and ASN of domain if available
rule	DGA rule
timeend	DGA validity stop (UTC+0)

3.3.3.3 Dataset size

Over 14.4 million DGA domains generated by 88 malware families were provided for testing/training.

3.3.4 SSL certificates collected from remote hosts visited by samples running in the sandbox

3.3.4.1 Description

This index contains information about SSL/TLS certificates observed in the sandbox for the set period. For example, a sample being executed in the sandbox makes an SSL/TLS connection to an external IP. Information on the certificate along with any meta-information is then collected. Note: many of these certificates may be for legitimate use even if it is a malware sample that is initiating the connection. Information on certificates may be used to enrich information from the DNS query index.

3.3.4.2 Index fields

Field name	Description
timestamp	timestamp when SSL/TLS cert was seen (UTC+0)
ip	queried IP where cert was seen
port	Port number of queried IP
asn	ASN of queried IP
geo	Geolocation of queried IP
subject_common_name	Server name covered by certificate
issuer_common_name	Certificate Authority issuing the certificate
sha1_fingerprint	SHA1 fingerprint of certificate
sha256_fingerprint	SHA256 fingerprint of certificate
sha512_fingerprint	SHA512 fingerprint of certificate
asn_name	Name of ASN of queried IP
cert_expiration	When the validity of the certificate expires
cert_issue_date	When the certificate was issued
cert_length	certificate length
cert_serial_number	serial number of the certificate
city	City where queried IP resides
ip4	IP address of queried device in decimal notation
issuer_country	Country of certificate Issuer
issuer_locality_name	Locality (eg. city) of certificate Issuer
issuer_organization_name	Organization name of certificate Issuer
issuer_organization_unit_name	Organization unit name (eg. department) of Issuer
issuer_state_or_province_name	State or province name of Issuer
key_algorithm	Encryption algorithm used for key exchange

latitude	Latitude of IP being queried (geolocation)
longitude	Longitude of IP being queried (geolocation)
md5	MD5 hash of sample initiating the query
naics	North American Industry Classification System (NAICS) code (for IP being queried)
protocol	Transport protocol used, ie. TCP or UDP
region	Region information of IP being queried (ie. additional geolocation data)
sample_timestamp	Timestamp of executed sample (UTC+0)
sector	Sector classification of IP being queried
sha1	SHA1 hash of sample initiating the query
sha256	SHA256 hash of sample initiating the query
sha512	SHA512 hash of sample initiating the query
sic	Standard Industrial Classification of IP being queried
signature_algorithm	hash algorithm used to sign the SSL certificate
ssl_version	SSL/TLS version being used
subject_country	Country of subject covered by the certificate
subject_locality_name	Locality of subject covered by the certificate
subject_organization_name	Organization covered by the certificate
subject_organization_unit_name	Organizational unit of Subject covered by the certificate
subject_state_or_province_name	Subject state or province

3.3.4.3 Dataset size

Over 68 000 records were provided for testing/training.

3.3.5 Known top domains

3.3.5.1 Description

This index provides a collection of known top domains obtained from various sources. It is intended to serve as a list of the most popular - presumably benign - domains. Note that TOP X lists change over time, hence the many more entries in this index. Additionally, sometimes malicious entries make such lists, as well as known domains become bad (hacked for example).

3.3.5.2 Index fields

Field name	Description
timestamp	Date of entry (UTC+0)
domain	Known domain added
position	Position on source list at time of entry
tag	set to `site-ranking`
source	Where known domain entry came from
resolv	GEO, IP and ASN of domain if available
timeend	Any time limited validity

3.3.5.3 Dataset size

Over 2 350 000 entries were provided for testing/training.

3.3.6 Further datasets

Further datasets have also been made available to aid the above process. These include JA3/JA3S[5] TLS fingerprints of connections and an AV detection dataset for malware hashes. A daily unregistered domain list of domains seen in sandbox queries (that do not match any known DGA) is also being shared. Other datasets will be explored in the future.

[5] https://github.com/salesforce/ja3

4 DGA Classification methods and Processing Pipeline

This section explains the DGA classification methods being initially proposed along with a processing pipeline that aims to achieve the SOCCRATES project goals.

4.1 Goals

The main goals detection wise of the classification methods being explored are as follows:

1. Identify new DGA domains in the DNS query dataset (different to DGA domains from known malware/seeds dataset).
2. Identify outliers in the DNS query dataset
3. Cluster similar query types in the DNS query dataset

In addition, the methods have to be scalable to be usable in production environments (specifically the Shadowserver sandbox environment).

4.2 DGA classification methods

The state of the art in DGA domain detection is characterised by two families of techniques: string similarity methods and deep learning methods. The former category is fairly old, whereas the latter is a more recent development.

Both families belong to the domain of supervised learning, although string similarity methods can also be used as a first step to later perform clustering (a form of unsupervised learning). Deep learning methods require large and well-structured training sets, whereas string-based methods can work with small training sets. Deep learning methods tend to generalize better and scale better, whereas string similarity methods will naturally overfit and scale worse (complexity can even be quadratic with some techniques).

4.2.1 String similarity methods

In string similarity methods, a character-based metric of similarity is established and used to compare strings. There are two main families of similarity metrics:

- edit based distance metrics
- token (or n-gram) based distance metrics

In edit-based distance metrics the amount of character edit operations to transform string A into string B is used as a measure of how similar the two strings are. There are many methods available to perform edit based similarity computation, with the Levenshtein distance[6] probably being the most popular. In

[6] https://en.wikipedia.org/wiki/Levenshtein_distance

token-based distance metrics strings A and B are broken in tokens or n-grams of a certain length of characters (typically n=2 or n=3), then two strings can be compared by comparing their token composition.

The main difference between these two methods is performance at scale. Although both methods grow quadratically in time complexity, token based methods can be greatly sped up by using linear algebra: strings can be represented as vectors of tokens and cosine distance can then be used to establish vector similarity (by measuring the angle between the vectors).

4.2.1.1 DGA classification with string similarity

String similarity approaches provide a straightforward methodology to classify DGA domains. If a domain is similar enough to a classified domain, we can infer the classification of the unlabelled domain from the classified one.

4.2.2 Machine Learning methods

A considerable amount of literature has been published on DGA detection algorithms based on machine learning techniques. These studies can be split in two categories according to the input data which they use: feature-based approaches and featureless approaches [4]. On the one hand, feature based approaches rely on extracting lexical or linguistic characteristics from the domain name such as median of n-grams, ratio of consecutive vowels or consonants or length of the top-level domain (TLD). Popular classifiers for this category include logistic regression and tree ensemble methods such as random forest algorithms [5][6]. In featureless approaches on the other hand, the domain name string is used directly as input to the algorithm either in the form of a sequence of characters or in combination with an additional encoding/tokenization step. Here, recurrent neural networks such as Long Short Term Memory (LSTM) [7][8] and convolutional neural networks (CNN) [9][10] have been shown to achieve high detection rates.

Studies introducing new DGA detection methods usually conduct both feature-based and featureless experiments and show that the latter obtains a better performance [8] [11]. Moreover, featureless approaches can be less computationally intensive since the pre-processing step of generating n-grams, or other string characteristics is not needed. For this reason, we opted in T4.2 for a featureless LSTM-based approach, which has the same architecture as that in [8] but a different choice of hyperparameters and size of latent space. This was necessary to adjust due to the structure of the datasets introduced in Section 3.

Whilst all approaches mentioned so far can perform both binary and family (multi-class) classification, we have restricted our focus to binary (DGA vs. non-DGA) detection. This was mainly since multi-class approaches generally fail at correctly labelling domains from a new malware family. Recently, effective methods for evading all the aforementioned DGA classifiers have been proposed in [12][13]. Given these limitations of deep-learning approaches, we have chosen the binary LSTM algorithm with the goal of rapid labelling/filtering of large amounts of domains, being fully aware that correct identification of new malware families can be better achieved by a complementary method such as the string grouper.

4.3 Software deliverables

Two software deliverables were implemented to perform supervised DGA classification. These deliverables are then deployed together to make the most out of their different advantages, improving the precision of the individual approaches.

4.3.1 String grouper DGA family classifier

The string grouper DGA family classifier relies on the string_grouper[7] python package to perform fast token-based similarity computation of strings.

The methodology is as follows:

1. given a labelled set composed of (domain, family),
2. for each unlabelled domain, we use the python string-grouper package to compute its similarity relative to all other domains in the labelled set,
3. then we retrieve the most similar,
4. if the similarity is high enough (compared to a precision threshold) the unlabelled domain is classified using the family of the most similar labelled domain.

Available parameters are as follows:

- token size: amount of characters in each token,
- min similarity: precision threshold. If no best match is found above this similarity threshold, the domain is not classified (family=None),
- bin size: it was observed that the precision of this methodology improves when comparing domains of similar size. This parameter determines the subset of the classified domain set that will be used to perform the comparison to the unlabelled domain. The subset is the result of filtering based on the domain length. If this parameter is set to 1, domains will be compared only to domains of the same length. If it is set to *n*, the unlabelled domain of length *k* will be compared to domains of length in the range *[k-n+1, k+n-1]* characters. If it set to 253 or higher, it will be compared to all domains available (253 is the largest character count allowed on a full domain name).

4.3.2 LSTM DGA binary classifier

As already mentioned before, our binary LSTM classifier is an adaption of [8]. The network structure is as follows:

- one embedding layer: it projects the sequences into a higher dimension space to allow for more degrees of freedom,
- one LSTM layer: it learns patterns of the characters in the sequences,
- output layer: produces the result for the given input (the predicted class based on some loss function) The output for the binary classification is either 0 (non-DGA) or 1 (DGA).

We opted for this rather small neural network architecture mainly due to computational performance reasons.

[7] https://github.com/Bergvca/string_grouper

The methodology is as follows:

1. Pre-processing steps
   The input data consists of domain names which are stripped of the top-level-domain, then mapped to a numerical encoding and padded with 0s such that all data samples have the same size. For this, we use the following characters dictionary: digits ['0', '1',...,'9'] mapped to integers 1-10, letters ['a', 'b',..., 'z'] mapped to integers 11-36 and symbols '-', '_', ' . ' mapped to integers [38-40]. Example: domain 'qeeyrag.com' becomes 'qeeyrag' mapped to array [27, 15, 15, 35, 28, 11, 17, 0, 0,.....0]. We take into consideration different maximum length limits (8,16,32,64,128 and 256 characters) to accommodate the diversity of domains in the datasets. Very long domain names (i.e., greater than 32 characters) occur rarely in the data but we need to have models that are prepared to handle domains up to 253 characters, which is the name length limit imposed on registering a new domain.

2. Training the LSTM model
   The binary LSTM model was trained using benign and malicious domains. This was a supervised approach: for every domain we provided the label to the model in order for it to be able to learn.

   Data used for testing and training description:

   - 2,096,038 benign domains (Alexa: 801,939, Cisco: 920,476, Quantcast: 373,623).
   - 1,244,715 unique DGA domains from 78 different families

   The split is shown in
   Table 1. It is worthwhile to mention that the benign set for training contains 30% Alexa, 40% Cisco and 30% Quantcast domains. Furthermore, the DGA training set consists of domains of 78 families selected from the 79 malware families in the FKIE dataset available at that point of time. One of the families contained only one domain so it was discarded. Moreover, domains from each malware family are present in both the training and test set, to ensure a good balance. We must remark however, that there is a strong class imbalance in the DGA domains: some malware families are represented by hundreds of thousands of domains, whilst some only have two or three domains. For binary classification this is not problematic, although it was an argument for not pursuing LSTM multi-class labelling here. The LSTM was trained for 5 epochs. We have also experimented with training it longer (25 epochs) but that has shown no significant improvement.

Table 1. Train/test split for the LSTM

Data	Benign domains	DGA domains
Train	1.000.000	1.003.642
Test	1.096.038	241.073
Total	2.096.038	1.244.715

3. Evaluation of the model
   Using the train/test split above, the performance of the model was evaluated. The main indicators we are interested in here are accuracy, precision and F1 score.

4. Labelling new domains
   Using the trained model, we can predict the class of unlabelled or new domains that have not been seen before.

4.3.3 DGA pipeline

If the deliverables are used together, the outcome of their classification can be judged using the following table:

Table 2. Classification matrix of initial DGA pipeline process

Classification	LSTM: DGA	LSTM: benign
String grouper: family x	tools agree. malicious domain, known family	Inconclusive: need human evaluation
String grouper: unknown	probably malicious, unknown family (possible new family)[8]	Inconclusive: need human evaluation
String grouper: whitelisted	LSTM false positive	tools agree. benign domain

[8] this set is the ideal candidate to perform clustering to identify new DGA families

5 Implementation and Deployment

This section explains the Docker-based DGA pipeline architecture introduced as part of this project. It also provides a description of the deployment process illustrated with screenshots.

5.1 DGA pipeline architecture

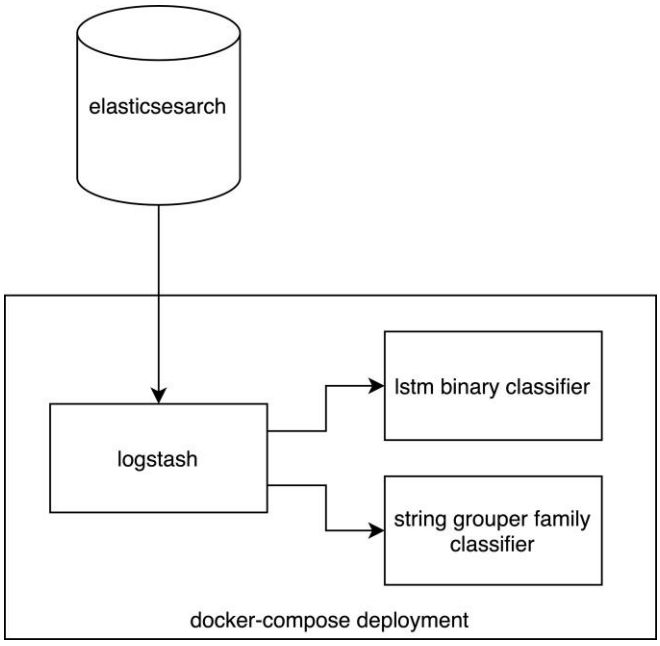

Figure 4. DGA pipeline architecture overview

The DGA pipeline architecture follows the microservices pattern. Both classifiers were developed in python. Microservices are implemented by packaging the classifiers in docker images, one for each container. The current architecture consists of three microservices:

- LSTM binary classifier: classifies a list of domains using the LSTM deliverable described in Section 4.3.2,
- string grouper family classifier: classifies a list of domains using the string grouper deliverable described in Section 4.3.1,
- Logstash: pulls the list of domains that need to be classified from the Shadowserver Elasticsearch repository to a csv file, which is then used by the two classifiers.

This microservice architecture has several advantages:

- consistency in deployment: Docker and docker-compose allow for deterministic implementation and deployment, see Section 5.3,
- modularity: the components are simple and can be used on their own,
- additional detectors/classifiers can be added in parallel with no need for changes in the implementation of the existing components. As an example, if in the future the data repository would change from Elasticsearch to a Postgres database, it would be sufficient to alter the logstash pipeline to ingest data from the Postgres (without having to alter the two classifiers).

- scalability: the number of instances of each microservice can be scaled up or down dynamically depending on the desired performance/infrastructure capabilities.

5.2 Implementation

Both classifiers have been implemented in the Python programming language. The string grouper DGA family classifier is built on top of the Python:3.8 official docker image[9] and it relies on the string_grouper project[10] for the string similarity computation. The LSTM binary classifier has been developed on top of the official tensorflow/tensorflow:2.2.0 docker image. Logstash[11] is a publicly available open source project.

5.3 Deployment

5.3.1 Current deployment

As mentioned in the previous subsections, the microservice pattern was implemented through docker containers. The deployment is performed by using the docker-compose project[12], which allows to deploy a number of containers together in one single script.

What follows is a sequence of screenshots that demonstrates the deployment of the DGA pipeline.

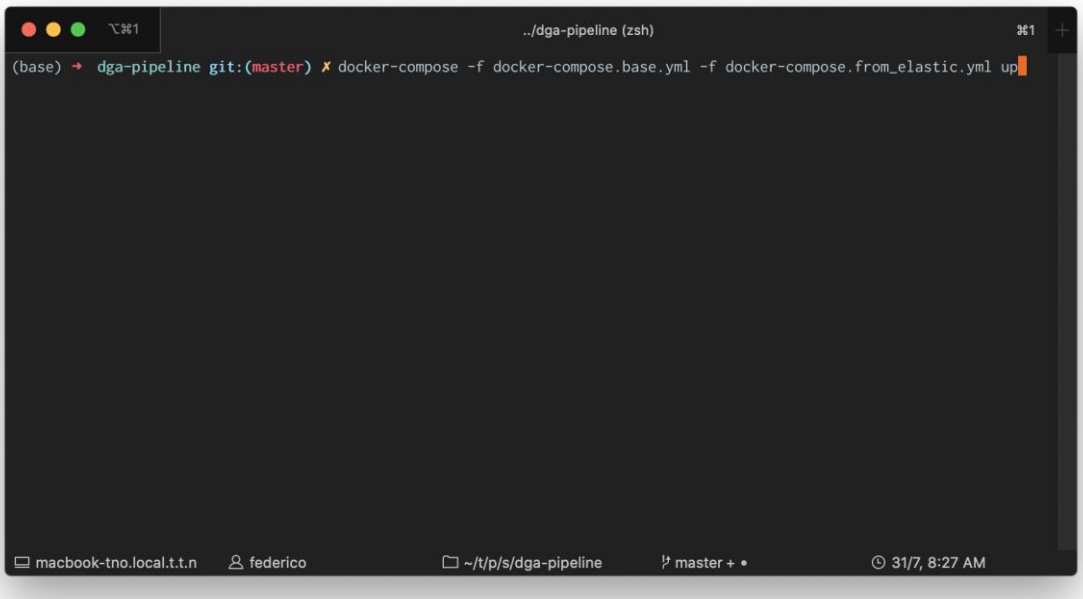

Figure 5. Docker-compose command to deploy the DGA pipeline

[9] https://hub.docker.com/_/python
[10] https://github.com/Bergvca/string_grouper
[11] https://www.elastic.co/logstash
[12] https://github.com/docker/compose

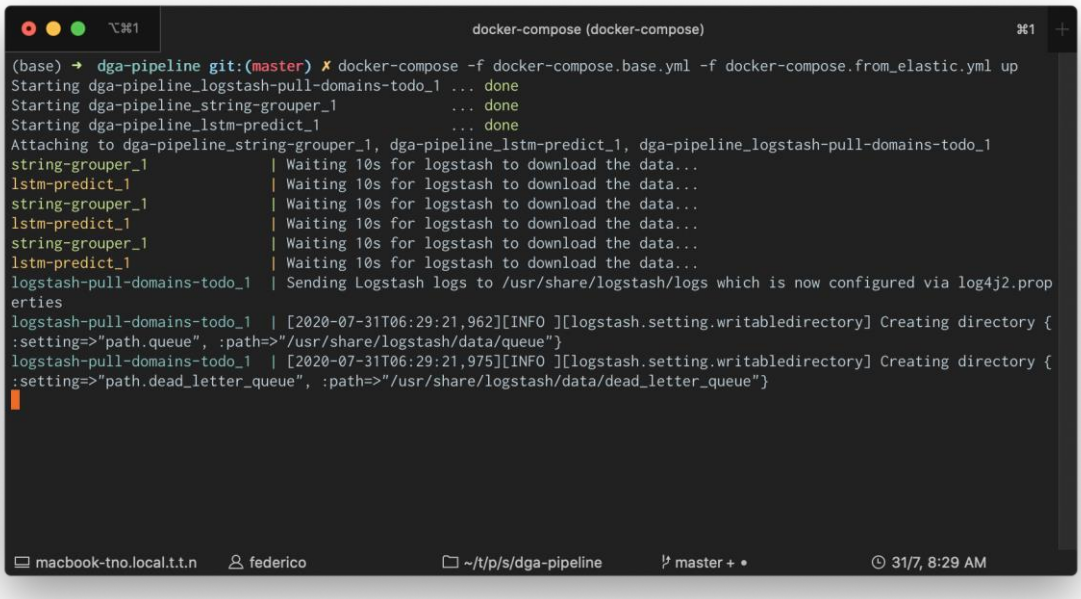

Figure 6. Start of the DGA pipeline deployment

In Figure 6 Logstash starts to download the list of domains to classify from Elasticsearch. String grouper and LSTM wait for logstash to finish.

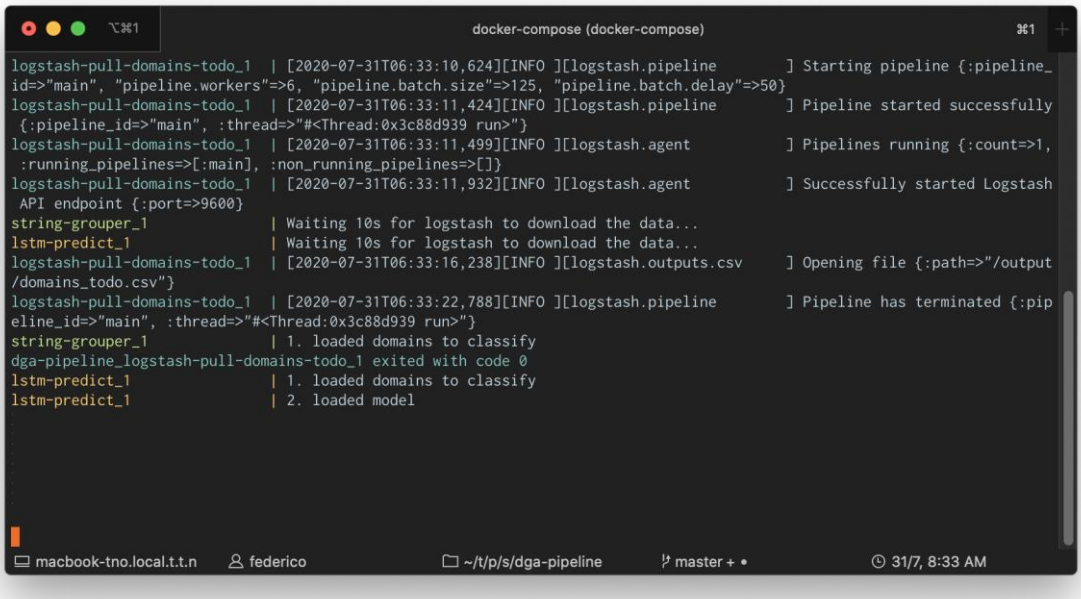

Figure 7. Logstash is finished, string grouper and LSTM start

Figure 8. The two classifiers output the results to stdout and, at the end, to a csv file

5.3.2 Production deployment

The production deployment will be mostly similar to the current deployment, but with three improvements:

- Single node scalability: Docker allows to deploy several instances (known as replicas) of each container image, making parallel computation on a single machine possible.
- Distributed scalability: Docker allows deployment over a cluster of computers through an orchestrator (e.g. Docker swarm, Kubernetes[13], etc). The architecture will be slightly adjusted to permit distributed deployment, making parallel computation over a cluster of computers possible.
- Output: as of now, the two classifiers output a csv file. In production, the content of these csv files will be uploaded to the Shadowserver repository.

[13] https://kubernetes.io/

6 Preliminary Results

The section discusses preliminary DGA classification results.

6.1 Performance evaluation

Preliminary results are encouraging. The LSTM binary classifier performs very well, with an overall precision of 96.5%. The string grouper family classification performance varies significantly between families. As of now, it looks like the best approach is to primarily rely on LSTM for the binary classification (DGA/benign) and then, if the domain is labelled as DGA, secondarily attempt family classification with string grouper. As expected, the string similarity approach has also worse run time performance (time) compared to the neural network approach. This is due to the fact that LSTM performs the model computation at a separate, previous, step, whereas string-grouper performs everything at run time.

6.2 String grouper evaluation

As mentioned in the previous paragraph, the performance of the family classification of the string grouper varies significantly (from poor to almost perfect) between families. Ideally, the domain to evaluate could be compared to the entire classified dataset available, but this is in practice computationally too intense for such a large number of domains. This is an inherent limitation of string similarity approaches. To improve computational performance, we use a random sample of 2000 domains from each family in the classified dataset instead of the whole dataset. For this reason, only families with at least 2000 domains could be included in the evaluation

The resulting classification performance can be seen in the table below. As expected, the classification precision shows an inverse correlation with regards to the number of domains in a given family. This follows from the fact that 2000 domains are a much larger proportion of domains in smaller families than it is in larger families.

The variability in performance between the families is so significant, with only 13 families reaching a precision above 80%, that string grouper will be used in the pipeline only to perform classification with regards to this subset of families. The list of families could be gradually expanded when better precision is reached for the other larger families. This could be achieved by increasing the random sample of 2000 domains per family to a higher amount specific to each family. The higher the string variability within a family the higher the number of domains that the random sample should contain.

Table 3. String grouper performance per anonymized malware family

Family	Domain count	Correct
1	439232	1.00
2	355528	0.28
3	291001	0.09

4	263281	0.42
5	201560	0.40
6	180264	0.80
7	143371	0.50
8	80000	0.40
9	72266	0.13
10	44292	0.27
11	32252	0.70
12	32020	0.11
13	30845	0.06
14	24549	0.87
15	21537	0.09
16	21420	0.13
17	20000	0.51
18	20000	0.08
19	18353	0.20
20	16363	0.98
21	15980	0.10
22	13056	0.17
23	10445	0.17
24	10140	0.75
25	9960	0.48
26	9789	0.81
27	8576	0.98
28	6000	0.66
29	6000	0.38
20	5989	0.17
31	5000	0.81
32	4400	1.00
33	4368	0.95
34	3348	0.84
35	3122	0.81
36	2704	0.95
37	2700	0.67

38	2600	0.46
39	2554	0.56
30	2106	0.88

6.3 LSTM binary classifier evaluation

On its own, the LSTM binary classifier performs well. The performance was evaluated on the available classified domains from the testing set shown in Table 4. The model was trained using exclusively the training set. Then, the model was used to classify the domains of the test set. As the actual classification of the domains in the test set is known, the performance of the classifier can be established by comparing the actual label with the predicted label.

Table 4. LSTM confusion matrix

	Correct classification	**Wrong classification**
DGA domains	17.4% (233,852) = True Positives	0.5% (7,221) = False Positives
not DGA domains	79.1% (1,058,430) = True Negatives	3% (37,603) = False Negatives
Total	96.5% (1,292,282)	3.5% (44,824)

A closer inspection of the false negatives revealed that many of these domain names were in fact strings created by concatenating proper words in English. This is a well-known limitation of the feature-less LSTM approach and can be tackled by an additional dictionary-based algorithm for instance. At the moment of writing this document, progress is being made into developing a method based on the WordGraph algorithm of [14] in order to enable detection of domain names which resemble English words. Although the initial step is focusing on the English language, it is worthwhile to mention that this approach needs to be enriched with foreign dictionaries to take into account different letter frequencies in different languages.

6.4 Future work

Future plans for scalable deployment are mentioned in Section 5.3.2. This will positively impact the computational performance for both classifiers. This may also be used as a work around to increase the classified examples that are provided to the string grouper. In practice, classifying *n* domains over 10 string grouper instances with 500 000 classified reference domains will take the same overall amount of time of classifying the same *n* domains over 1 string grouper with 50 000 classified reference domains, but with a significant improvement of classification performance.

In order to make distributed deployment possible, the architecture will have to be altered by adopting a queue system to distribute the domains to classify over the classifiers.

A third software deliverable will be added to the DGA pipeline to perform discovery of unknown families. As mentioned in Section 5.2, string similarity techniques can be used as a first step for clustering - a form of unsupervised learning where classes of similar objects are discovered in an unclassified dataset. String grouper classifies domains as belonging to a known DGA family, or not. If enough domains are classified as malicious by LSTM but not belonging to any known family by the string grouper, it could be that there is a new DGA family in the dataset. String grouper can be used to compute the pairwise similarity matrix of a list of domains. Feeding the similarity matrix of the unknown family domains to a clustering technique may lead to the discovery of new families if there are enough related domains.

Finally, the software developed in this work package could be eventually integrated to the battery of detectors that is deployed in SOCCRATES WP4.

7 Conclusion and Next Steps

The primary lesson learnt so far is that it is easier to combine a series of specialized DGA detectors rather than focus on designing a perfect one. Providing a set of tools in a framework that enables easy addition or disabling of individual tools seems to be a reasonable path. This may also aid in the challenge of achieving the scalability and performance needed to deal with potentially millions of domains. In some cases, it may be more feasible to utilize various heuristics to minimize datasets when possible, especially in scenarios that require relatively fast classification. For example, focusing on domains that do not exist – i.e. return an NXDOMAIN response to a resolver query on a network. Such responses often accompany DGA malware presence, as by definition such malware will try to connect to multiple domains if a DGA is activated, and multiple of those may not actually be registered. This may be of significance if algorithms developed here are integrated with network detection tools being developed as part of Task 4.1, where typically detection times will play a more critical role, as these tools are going to be deployed directly on SOCCRATES Pilot networks.

Utilizing Docker also proved to be a successful approach, enabling easier testing both on the training/validation datasets made available and on additional data on the Shadowserver backend.

Future work in Task 4.2 is envisioned as follows:

- improve string grouper precision (see Section 6.1.1),
- add clustering component for unknown families (Section 6.2),
- add dictionary-DGA detection component (Section 6.2),
- production grade scalability and performance for all components (Section 5.3.2).

Furthermore, work here and subsequently in WP6 and WP7 will focus on the integration/testing of software components in SOCCRATES, including:

- regular (automated) deployment of DGA pipeline within Shadowserver infrastructure,
- upload/update and storage of DGA detection results in a Shadowserver data repository,
- development of API(s) to the SOCCRATES TIP enabling the DGA lookup of domains (including a family lookup) as well as other potential features as described in D2.2 “System Architecture and Interface Specifications (Initial)”.

8 References

[1] Gameover P2P, Malpedia entry, https://malpedia.caad.fkie.fraunhofer.de/details/win.gameover_p2p [retrieved August 2020]

[2] Cryptolocker, Malpedia entry, https://malpedia.caad.fkie.fraunhofer.de/details/win.cryptolocker [retrieved August 2020]

[3] Avalanche Network Dismantled in International Operation, Europol https://www.europol.europa.eu/newsroom/news/%E2%80%98avalanche%E2%80%99-network-dismantled-in-international-cyber-operation [retrieved August 2020]

[4] Sivaguru, Raaghavi et al. "An Evaluation of DGA Classifiers." *2018 IEEE International Conference on Big Data (Big Data)* (2018): 5058-5067.

[5] Yang, Luhui et al. "Detecting Word-Based Algorithmically Generated Domains Using Semantic Analysis." *Symmetry* 11.2 (2019): 176.

[6] Schüppen, Samuel et al. "FANCI : Feature-based Automated NXDomain Classification and Intelligence." *USENIX Security Symposium* (2018).

[7] Tran, Duc et al. "A LSTM based framework for handling multiclass imbalance in DGA botnet detection." *Neurocomputing* 275 (2018): 2401-2413.

[8] Woodbridge, Jonathan et al. "Predicting Domain Generation Algorithms with Long Short-Term Memory Networks." *ArXiv* abs/1611.00791 (2016)

[9] Yu, Bin et al. "Inline DGA Detection with Deep Networks." *2017 IEEE International Conference on Data Mining Workshops (ICDMW)* (2017): 683-692.

[10] Zhou, Shaofang et al. "CNN-based DGA Detection with High Coverage." *2019 IEEE International Conference on Intelligence and Security Informatics (ISI)* (2019): 62-67.

[11] Lison, Pierre and Vasileios Mavroeidis. "Automatic Detection of Malware-Generated Domains with Recurrent Neural Models." *ArXiv* abs/1709.07102 (2017)

[12] Peck, Jonathan et al. "CharBot: A Simple and Effective Method for Evading DGA Classifiers." IEEE Access 7 (2019): 91759-91771.

[13] Yun, Xiaochun et al. "Khaos: An Adversarial Neural Network DGA With High Anti-Detection Ability." *IEEE Transactions on Information Forensics and Security* 15 (2020): 2225-2240.

[14] Pereira, Mayana et al. "Dictionary Extraction and Detection of Algorithmically Generated Domain Names in Passive DNS Traffic." *RAID* (2018).

9 Abbreviations

This is a standard list with abbreviations used for all technical SOCCRATES deliverables

Acronym	Description
ACT	semi-Automated Cyber Threat intelligence
ADG	Attack Defence Graph
AEF	Argus Event Format
AI	Artificial Intelligence
AIT	AIT Austrian Institute of technology
API	Application Programming Interface
APT	Advance Persistent Threat
ASN	Autonomous System Number
ATOS	ATOS Spain
AV	AntiVirus
BPMN	Business Process Model and Notation
C2	Command and Control
CC	Command and Control
CERT	Computer Emergency Response Team
CMDB	Configuration Management Database
CSIRT	Computer Security Incident Response Team
CoA	Course of Action
CTI	Cyber Threat Intelligence
DC	Data Center
DGA	Domain Generated Algorithm
DNS	Domain Name System
EDR	Endpoint Detection and Response
ELK	Elasticsearch/Logstash/Kibana
FRS	Foreseeti
FSC	F-secure
ICT	Information and Communication Technology
IDS	Intrusion Detection System
IMC	Infrastructure Modelling Component
IMT	Institut Mines Télécom – Télécom SudParis
INTF	Interface
IoC	Indicators of Compromise
IP	Internet Protocol
IPS	Intrusion Prevention System
IRM	Incident Response and Management
ITIL	Information Technology Infrastructure Library
KTH	Kungliga Tekniska högskolan - Royal Institute of Technology
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol

LSTM	Long Short Term Memory (neural network)
M_n	Infrastructure Model (at time *n*)
MNM	mnemonic
MSSP	Managed Security Service Provider
MTTD	Mean Time To Detection
NOC	Network Operations Centre
OT	Operational Technology
OS	Operating System
RORI	Return on Response Investment
SDN	Software Defined Network
SHS	Shadowserver
SIEM	Security information and event management
SOAR	Security Orchestration, Automation and Response
SOC	Security Operation Centre
SOCCRATES	SOC & CSIRT Response to Attacks & Threats based on attack defence graph Evaluation Systems
SSL	Secure Sockets Layer
TAP	Test Access Point
TI	Threat Intelligence
TIP	Threat Intelligence platform
TLD	Top Level Domain
TLS	Transport Layer Security
TNO	Nederlandse Organisatie voor toegepast natuurwetenschappelijk onderzoek
TTC	Time To Compromise
UC	Use Case
VLAN	Virtual LAN
VM	Virtual Machine
VTF	Vattenfall