

The Open University of Israel

Department of Mathematics and Computer Science

Cyber-Attacks Discovery via Analysis of DNS

Project report submitted as partial fulfillment of the requirements

Towards M.Sc. degree in Computer Science

The Open University of Israel

Computer Science Division

By

Eyal Paz

Prepared under the supervision of **Prof. Ehud Gudes**

January 2020

Table of Content

Table of Content	2
List of Figures	4
List of Tables	5
Abstract	6
1. Introduction	7
2. Project Description.....	8
2.1 Project Goals	8
2.2 Project Importance	8
3. Implementation	9
3.1 Development Workspace.....	9
3.1.1 Operating System.....	9
3.1.2 Development Languages and Frameworks.....	9
3.1.3 Source Control	11
3.2 Architecture	11
3.2.1 Data Collection	11
3.2.2 Data Enrichment	14
3.2.3 Training Models	17
3.2.4 Operational Phase.....	19
3.3 Domain Classifiers	21
3.3.1 Social Network Analysis over Domain-IP Relationships	21
3.3.2 Machine Learning Blacklisting at Time-of-Registration	25
3.3.3 Predictive Blacklisting	30
4. Experiment	34

4.1 Data Cleaning	34
4.2 Data Separation	35
4.3 Evaluation	36
5. Project Configurations and Operation	39
5.1 Setup	39
5.1.1 Software Components	39
5.1.2 Data Source Configurations	39
5.1.3 Logging	40
5.2 Running the Backend	40
5.3 Running the Frontend	42
5.4 Frontend Operation	42
5.4.1 Domain Reputation	42
5.4.2 Explore Dataset	44
5.4.3 Data Schema	44
5.4.4 Train Model	45
6. Summary and Conclusions	46
References	47
Appendix A – Example of Benign Domains Classification Result	48
Appendix B – Example of Malicious Domains Classification Result	49

List of Figures

Figure 1 – on the left box are the top 15 rows of Alexa ranking feed and on the right box Cisco Umbrella ranking feed on both snapshots were taken on 17-Dec-2019	12
Figure 2 - DataSource class diagram	14
Figure 3 – Initial data collection flow.....	16
Figure 4 - Model class diagram	18
Figure 5 - Training flow	19
Figure 6 - domains table SQL CREATE query.....	20
Figure 7 - Operational system flow.....	21
Figure 8 - A topology-based flow model for computing domain reputation [1] architecture.....	22
Figure 9 - Append enriched domain record to the graph function	23
Figure 10 - Ego graph with radius 3 of edition.cnn.com.....	24
Figure 11 - SNA model prediction process.....	25
Figure 12 - A high-level overview of PREDATOR [3] architecture.....	26
Figure 13: Proactive Malicious Domain Name Discovery System [4] architecture	31
Figure 14 – on the left word statistics of the word “free”, and on the right word statistics for the word “pay”	32
Figure 15 - Markov model domain names generator	33
Figure 16 - Creating a blacklist of predicted malicious domains	33
Figure 17 - Classification data separation charts. Blue represents benign sample probabilities, and red are malicious sample probabilities.....	35
Figure 18 - ROC Curve using the threshold shown in table 9	37
Figure 19 - ROC of PREDATOR [3] The inlay figure shows the ROC curve under the range of 0–5% false positives.....	38
Figure 20 - fetch-domains.py manual	40
Figure 21 - fetch-domains.py manual	41
Figure 22 - model_manager.py manual.....	41
Figure 23 - app.py manual	42

Figure 24 - "Domain Reputation" screenshot, input box filled with the URL https://web.whatsapp.com/.....	42
Figure 25 - Domain reputation result for the URL https://web.whatsapp.com/	43
Figure 26 - SNA Ego graph with radius 2 for the domain web.whatsapp.com.....	43
Figure 27 – “Explore Dataset” screenshot.....	44
Figure 28 – “Data Schema” screenshot	44
Figure 29 – “Train Model” screenshot.....	45
Figure 30 - example output of "Train Model"	45

List of Tables

Table 1 – Summary of the data sources used on this project	13
Table 2 - The enriched record of edition.cnn.com	15
Table 3 - a snippet of benign domains taken from the database.....	19
Table 4 - a snippet of malicious domains taken from the database	20
Table 5: Summary of PREDATOR [3] features, each feature is categorical, continuous or ordinal.	27
Table 6 - Ranking of feature importance in PREDATOR [3] (D for domain profile category, R for registration history category, and B for batch correlation category).	28
Table 7 - Records from the dataset for building an example model for PayPal phishing detection.....	29
Table 8 - Decoded features for the records of table 7.	29
Table 9 - The selected threshold for the classifiers	36
Table 10 - Confusion matrixes of the classifiers’ evolution	36
Table 11 - PREDATOR [3] detection rates under a 0.35% false positive rate	38
Table 12 - Classifier detection rate under a 0.35% false positive rate	38
Table 13 - Example of the benign domains classification result.....	48
Table 14 - Example of the malicious domains classification result	49

Abstract

The Domain Name System (DNS) is an essential component of the internet infrastructure that translates domain names into IP addresses. Threat actors abuse that system by registering and taking over of thousands of Internet domains every day to launch cyber-attacks, such as spam, phishing, botnets, and drive-by downloads. The main solution to counteract this threat is currently reactive blacklisting. Since cyber-attacks are mainly performed over short periods of time, reactive methods are too slow and ineffective. As a result, new approaches to early identification of malicious websites are needed. In the last ten years, many novel papers were published offering a system that calculates domain reputation for suspected domains that are not listed in a common black-list list. This project implements three different approaches and evaluates their effectiveness in detecting malicious domains. The approach that outperforms the others in the project's experiments was social network analysis, it achieved a 60.71% detection rate with a false positive rate of 0.35%.

1. Introduction

In current days, information security is an important aspect of any organization's business. Finding a cyber-attack in an enterprise network is often analogous to finding the needle in the haystack. Analysis of DNS traffic can be helpful to that end. Providing high quality, cheap and fast attack detection technique.

Information security usually comes with three price tags, network performance impact, privacy violation, and false positive alerts. Network performance impact cause due to deep traffic inspection, privacy violation due to the need to decrypt private encrypted traffic for inspection, and false positive alerts due to the variance of each network.

Attack discovery by analysis of DNS traffic, reduce the price tag of all three aspects. The reason for that is because DNS is a very simple plaintext protocol containing short messages, usually over UDP protocol. Therefore, its analysis is much simpler and faster, however, its true positive detection would always be a subset of the detection that can be made by full packet inspection.

This project covers three major techniques of discovering cyber-attacks via analysis of DNS: passive DNS analysis, domain registration WHOIS record analysis, and predictive domain names blacklisting. To cover as much ground, each paper selected on this project has taken not only a different data type input but also a different solution approach as well: social network analysis, machine learning, and Markov chain model. The project provides a working system for detecting spam and phishing domains based on novel academic research.

2. Project Description

This section elaborates on the project goals and Importance.

2.1 Project Goals

The project's main goal is to implement three techniques of cyber-attacks discovery via analysis of DNS data: passive/active DNS analysis, WHOIS domain records analysis, and purely strings based analysis predictive blacklisting.

a secondary goal is to create a dataset of benign and malicious domains that may be used in other research projects or to be used as a benchmark for domain classifiers.

another secondary goal is making the classifiers available for anyone for free. That means that the classifier should be based only on open repositories and not contain a feature that can be extracted from non-publicly free available data sources.

2.2 Project Importance

There are three key advantages in DNS analysis for cyber-attacks discovery:

1. Relatively cheap, in comparison to other approaches e.g. deep packet inspection
2. DNS is plaintext, avoid problems of encrypted traffic inspection
3. Privacy-preserving, encrypted traffic doesn't need to be deciphered

There are numerous academic papers offering detection algorithms for cyber-attacks discovery by DNS analysis. However, none of them offers an open-source implementation or give access to the dataset they used to evaluate the novel approach. Therefore, it's impossible to run a true comparison between the different papers and approaches. The project implementation could help to achieve this kind of comparison. By releasing an open-source classifiers implementation, a dataset, and the dataset creation source code.

3. Implementation

This section covers the implementation details of the “Cyber-Attacks Discovery via Analysis of DNS” project including system components, design, architecture, frameworks, development workspace, and technologies.

3.1 Development Workspace

This section covers the development’s workspace and technologies used in the project implementation process.

3.1.1 Operating System

The project was developed and tested on an Ubuntu 18.04 OS. Since all the project technologies that are mentioned in the next section are cross-platform, it can run on other OS such as Windows as well.

3.1.2 Development Languages and Frameworks

The project database used for storing the dataset was Postgres SQL. The Database management was made with “pgAdmin” – a Python Web UI for Postgres SQL.

Redis was used for internal application caching and asynchronous messaging queue. Redis management was made with “Redis-commander” – a Node Web UI for Redis.

The programming language used in the project was Python. In the algorithm development phase of the project, “Jupyter Notebook” was used. After the algorithms were implemented and fine-tuned, I switch to Visual Studio Code for wrapping up the project.

Other than the standard libraries, it’s worth mentioning the usage of the following open source packages which saved much work in the implementation process:

- **Numpy** - the fundamental package for scientific computing with Python. It contains among other things: a powerful N-dimensional array object, useful linear algebra, Fourier transform, and random number capabilities.
- **Pandas** - flexible, and expressive data structures designed to make working with structured and time series data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real-world data

analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open-source data analysis/manipulation tool available in any language.

- **SQLAlchemy** – a Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL. SQLAlchemy provides a full suite of well-known enterprise-level persistence patterns, designed for efficient and high-performing database access, adapted into a simple and Pythonic domain language.
- **Scikit-Learn** – a Python package for machine learning built on top of SciPy and distributed under the 3-Clause BSD license. Scikit-Learn is compatible to work with Numpy and Pandas mentioned above packages.
- **XGBoost** – a Python optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. The same code runs on the major distributed environment (Hadoop, SGE, MPI) and can solve problems beyond billions of examples. XGBoost is compatible with Scikit-Learn classifier APIs.
- **NetworkX** – a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. The SNA classifier in this work is built on top of this package.
- **Matplotlib** – a Python 2D plotting package that produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits.
- **Compound-Word-Splitter** – a python natural processing language (NLP) package that, splits words that are not recognized by dictionary whitelists such as spell checkers into the largest possible compounds.

- **Redis (Python package)** – The Python interface to the Redis key-value store.
- **Pyreverse** – a set of utilities to reverse engineer Python code. It uses a representation of a Python project in a class hierarchy which can be used to extract any information such as generating UML diagrams. The class diagram shown in figures 2 and 4 were drawn using Pyreverse
- **Pyasn** – a Python extension module that enables very fast IP address to Autonomous System Number (ASN) lookups. Current state and Historical lookups can be done, based on the MRT/RIB BGP archive used as input.
- **Flask** – Flask is a lightweight web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications

3.1.3 Source Control

During development, the project was hosted on GitHub as a private repository. After completion, it's now open as an opensource project as a public repository. The project public repository is <https://github.com/eyalsus/domain-classifier>.

3.2 Architecture

This section covers the architecture description of data collection, training model and the system operational phase.

3.2.1 Data Collection

Data for this project was gathered from 4 free origins: Cisco Umbrella 1 Million popular DNS records and Alexa top 1 Million popular sites were used for benign domain collection, OpenPhish and PhishTank were used for malicious domains collection. all the mentioned origin publishes a CSV file which updates at least daily.

Alexa dataset is designed to be an estimate of a website's popularity. As of May 2018, Alexa claims the ranking is calculated from a combination of daily visitors and pageviews on a website over a 3-month period.

Cisco Umbrella, formally known as OpenDNS, the dataset is based on the Umbrella global network of more than 100 Billion DNS queries per day, across 65 million unique

active users, in more than 165 countries. Although the data source is quite different from Alexa's, it's arguably considered to be more accurate as it's not based on only HTTP requests from users with browser additions.

Figure 1 shows the clear difference between the feeds. For example, while Netflix owns 10 out of the top 15 domains in the Cisco Umbrella ranking the first entry of a Netflix domain in Alexa ranking on the same day is at the rank of 22. Another great example are Microsoft's Windows updates domains that have 3 out of the top 15 domains in Cisco Umbrella ranking but get much lower ranks on Alexa ranking.

1, google.com	1, netflix.com
2, youtube.com	2, api-global.netflix.com
3, tmall.com	3, prod.netflix.com
4, baidu.com	4, push.prod.netflix.com
5, qq.com	5, ftl.netflix.com
6, sohu.com	6, prod.ftl.netflix.com
7, facebook.com	7, ichnaea.netflix.com
8, taobao.com	8, nrdp.prod.ftl.netflix.com
9, login.tmall.com	9, google.com
10, yahoo.com	10, secure.netflix.com
11, jd.com	11, microsoft.com
12, amazon.com	12, nrdp51-appboot.netflix.com
13, wikipedia.org	13, windowsupdate.com
14, 360.cn	14, ctldl.windowsupdate.com
15, sina.com.cn	15, data.microsoft.com

Figure 1 – on the left box are the top 15 rows of Alexa ranking feed and on the right box Cisco Umbrella ranking feed on both snapshots were taken on 17-Dec-2019

OpenPhish and PhishTank dataset are based on community trusted members who share their threat intelligence of phishing websites. It's interesting to know that PhishTank was founded by OpenDNS as a by the community which several years later released the Cisco Umbrella feed as well. Unlike benign domain sources, the malicious domain sources contain URLs and not domains. Therefore before adding them to the dataset, some parsing should be made to extract the domains. Table 1 summarizes the data sources' characteristics.

	Cisco Umbrella (OpenDNS)	Alexa	OpenPhish	PhishTank
Classification	Benign	Benign	Malicious	Malicious
Update	Daily	Daily	Hourly	Hourly
Records Type	Domain	Domain	URL	URL
License	Free	Free	Free for partial content, paid for full feed access	Free, but registration required
Source	DNS queries	Web page views	Community	Community
Further Context	Popularity rank	Popularity rank	Phishing Target available on a paid subscription	Phishing Target
Est.	2016	1996	2014	2006
Feed URL	http://s3-us-west-1.amazonaws.com/umbrella-static/top-1m.csv.zip	http://s3.amazonaws.com/alexa-static/top-1m.csv.zip	https://openphish.com/feed.txt	http://data.phishtank.com/data/online-valid.csv

Table 1 – Summary of the data sources used on this project

Figure 2 describes the class diagram of the DataSource classes. Notice that there is no dedicated class for Cisco Umbrella, that is because Cisco Umbrella feed mimics the conventions laid out by Alexa veteran feed as can be seen in figures 1.

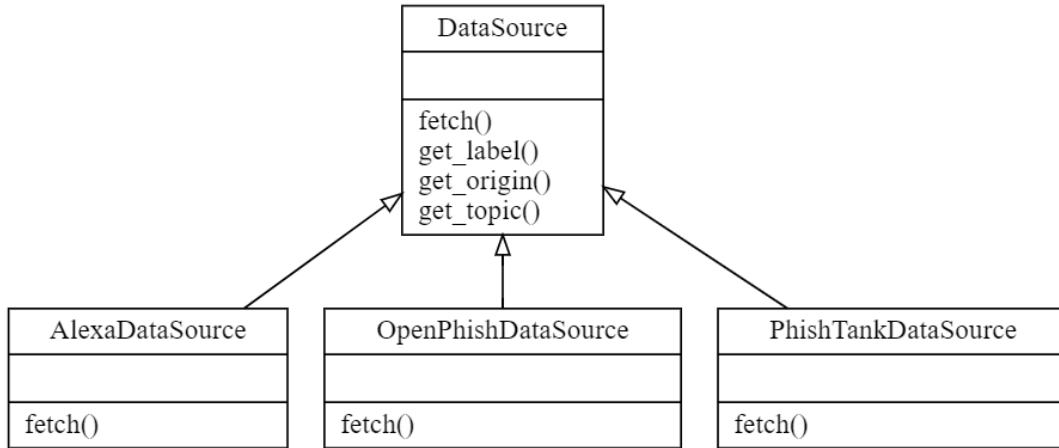


Figure 2 - DataSource class diagram

3.2.2 Data Enrichment

The data collected from the sources described in the previous section is enriched with DNS “A” record, DNS “NS” record, Autonomous System (AS) data, and parsed for processing convenience. DNS “A” record is the IPv4 address of the queried domain, DNS “NS” record is the nameserver of the queried domain. AS is the upper hierarchy of the IP address. The record after enrichment contains the following fields:

domain – the input domain

label – ‘0’ if the domain is benign, ‘1’ if the domain is malicious.

timestamp – the timestamp if the first time the data collection encountered the domain.

base_domain – the domain as it appears on the whois registration, for example, edition.cnn.com base domain is cnn.com.

domain_name – the base domain name without the public suffix, for example, edition.cnn.com domain name is cnn.

domain_ip – the current domain’s DNS “A” record, IP address of the input domain.

as_number – Autonomous System (AS) number of the domain’s IP address.

as_subnet – the matching subnet of the IP address’s Autonomous System Number (ASN).

as_name – the official name of the ASN owner of the domain IP address.

nameserver – the current base domain DNS “NS” record, nameserver of the base domain.

ns_base_domain – the base domain of the nameserver.

ns_domain_ip – the current domain’s DNS “A” record, IP address of the nameserver.

ns_as_number – ASN of the nameserver’s IP address.

ns_as_subnet – the matching subnet of the nameserver’s IP address ASN.

ns_as_name – the official name of the nameserver’s IP address ASN owner.

an example of an enriched record is shown in table 2.

domain	edition.cnn.com
timestamp	2020-01-13T19:36:02.817160
base_domain	cnn.com
domain_name	cnn
domain_ip	151.101.65.67
as_number	54113
as_subnet	151.101.64.0/22
as_name	FASTLY - Fastly, US
nameserver	ns-1630.awsdns-11.co.uk
ns_base_domain	awsdns-11.co.uk
ns_domain_ip	205.251.198.94
ns_as_number	16509
ns_as_subnet	205.251.198.0/24
ns_as_name	AMAZON-02 - Amazon.com, Inc., US

Table 2 - The enriched record of edition.cnn.com

The initial data collection phase flow, described in Figure 3, contained the following steps:

1. Fetch feeds request new data from a concrete data source
2. The data source fetches a feed from domains/URLs from its concrete vendor

3. Fetch feeds publish a message with the domain name to the “New URLs” messaging queue
4. Enrich domain reads messages from the “New URLs” messaging queue.
5. Enrich domain enriches the domain with DNS, IP, ASN and nameservers data the publish the enriched record to “Enriched Domain” queue.
6. The model manager reads messages from the “Enriched Domain” messaging queue.
7. Model manager commands the database connector to persist the enriched records
8. The database connector manages the interaction with the Database and persists the enriched records in the Database.

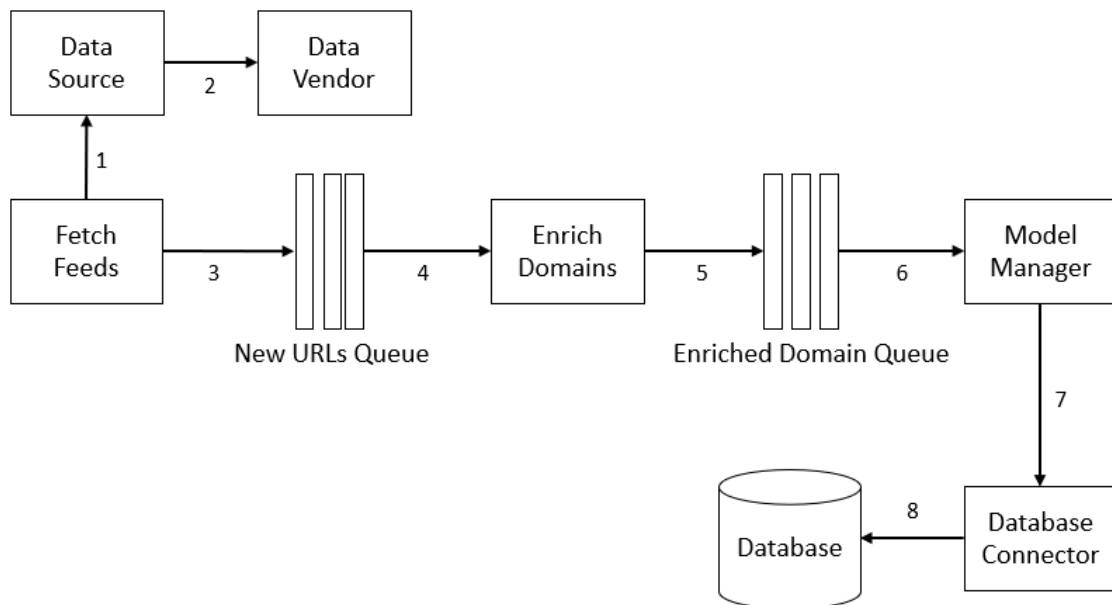


Figure 3 – Initial data collection flow

3.2.3 Training Models

The collected data is used for training the domain classifier models. Figure 4 describes the class diagram of the models training phase. The below classes were developed as a part of this project:

Model – an abstract class which define the Model interface

AggregatorModel – this class employs the composite pattern. It extends the Model class and contains a list of Model objects.

MLModel – extends Model class. Support all Scikit-Learn classifiers and any other API complaint package, e.g. XGBClassifier from XGBoost package.

MarkovModel – extends Model class. The class is composed of the opensource MarkovChain class, and leverage compound-word-splitter NLP package.

SnaModel – extends Model class. The class is composed of the opensource Graph class from the NetworkX package. Graph class was leveraged to implement the SNA classifier.

MarkovChain – Base on an open-source implementation of Markov Chain, but was extended to the purpose of this project.

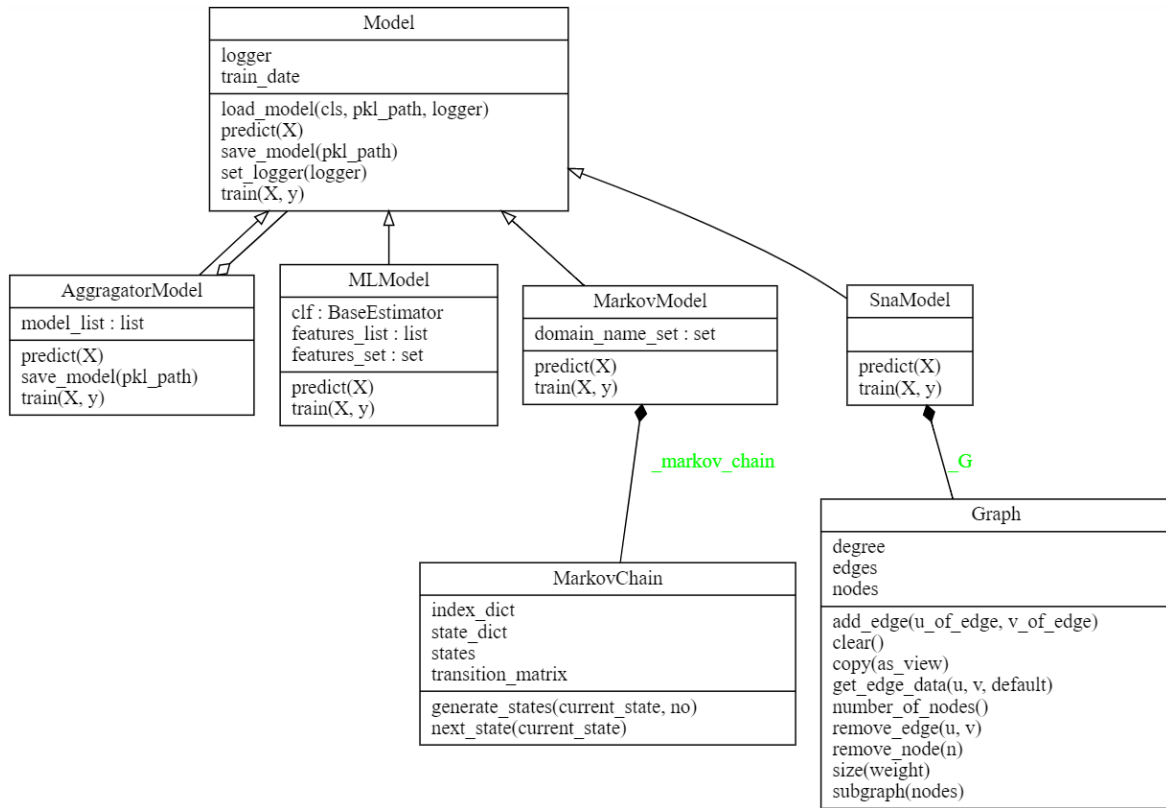


Figure 4 - Model class diagram

The training phase flow, described in Figure 5, contained the following steps:

1. Model manager fetches enriched domain records by calling the Database connector module
2. The database connector fetches the records from the Database
3. The model manager relies on the records to the aggregator model and commands it the begin the training.
4. The aggregator model trains the models it currently contains.
5. The model is serialized and saved to disk for persistent storage.

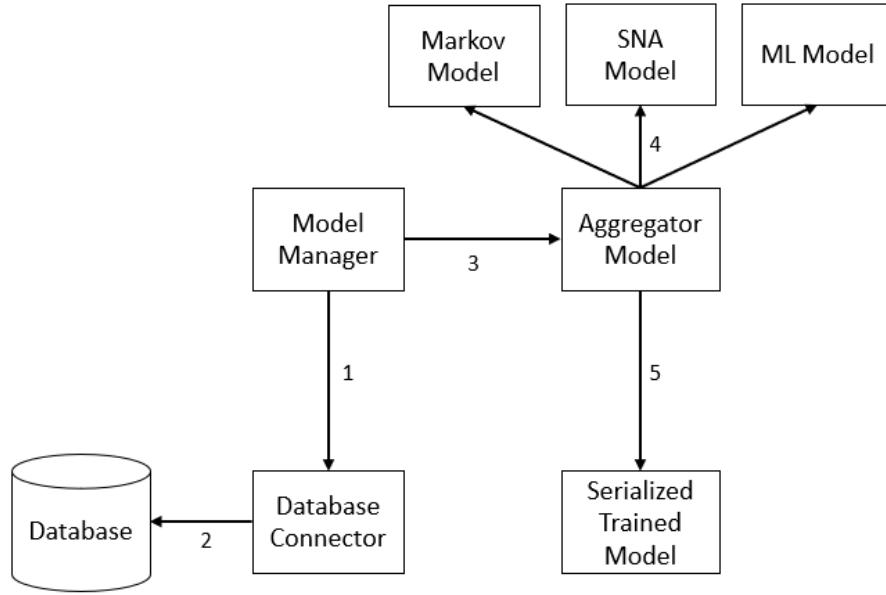


Figure 5 - Training flow

3.2.4 Operational Phase

After a significant amount of data was collected in the flow described in section 3.2.1 and the models were trained in the flow described in section 3.2.2 the system is now in the operational phase. The operational phase integrates both previous flows and extends them. On top of the enriched records, the model aggregator feeds the DB with the current models' verdict. This way the model evaluation is constantly being made versus the most recent ground-truth dataset. The models' aggregator is scheduled to retrain the models daily. Tables 3 and 4 show a snippet of the dataset, a larger part of the dataset is available on appendix A.

domain	label	timestamp	domain_ip	as_number	nameserver	sname	train_date
netice.az	0	12/17/19 10:09 AM	104.27.149.152	13335	gina.ns.cloudflare.com	0.331968	12/17/19 10:47 AM
iqtds.com	0	12/17/19 10:09 AM	93.174.95.2	202425	ns1.reg.ru	0.47648	12/17/19 10:47 AM
orissimo.it	0	12/17/19 10:09 AM	104.20.5.243	13335	jule.ns.cloudflare.com	0.255366	12/17/19 10:47 AM
mutisite.com	0	12/17/19 10:09 AM	104.31.72.253	13335	adrian.ns.cloudflare.com	0.311023	12/17/19 10:47 AM
otoy.com	0	12/17/19 10:09 AM	104.20.40.12	13335	bart.ns.cloudflare.com	0.247716	12/17/19 10:47 AM

Table 3 - a snippet of benign domains taken from the database

domain	label	timestamp	domain_ip	as_number	nameserver	sname	train_date
taarefeahlal baitam.com	1	12/17/19 9:55 AM	160.153.137.163	26496	ns24.domaincontrol.com	0.786192	12/17/19 8:47 AM
www.britishairportcars.co.uk	1	12/17/19 9:55 AM	35.237.67.68	15169	ns2.360expose.com	0.635082	12/17/19 8:47 AM
www.manawikassanstha.com	1	12/17/19 9:11 AM	69.175.87.74	32475	ns111.webhostingworld.net	0.694392	12/17/19 8:47 AM
silvanoyjairo.webcindario.com	1	12/17/19 9:11 AM	5.57.226.202	29119	ns-cloud-d3.googledomains.com	0.813255	12/17/19 8:47 AM
secure.runescape.com-ms.xyz	1	12/17/19 9:11 AM	23.254.225.128	54290	ns28.domaincontrol.com	0.800725	12/17/19 8:47 AM

Table 4 - a snippet of malicious domains taken from the database

The complete database table contains more fields than shown in tables 3-4, the table definition is shown in figure 6.

```
CREATE TABLE domains
(
    domain text,
    label bigint,
    "timestamp" timestamp without time zone,
    base_domain text,
    domain_name text,
    domain_ip text,
    as_number text,
    as_subnet text,
    as_name text,
    nameserver text,
    ns_base_domain text,
    ns_domain_ip text,
    ns_as_number text,
    ns_as_subnet text,
    ns_as_name text,
    markovmodel double precision,
    sname double precision,
    mlmodel_xgbclassifier double precision,
    mlmodel_logisticregression double precision
);
```

Figure 6 - domains table SQL CREATE query

The operational system phase flow, described in Figure 7, contained the following steps:

Steps 1-6 are the same as described in the data collection section 3.2.1 and figure 3.

7. The model manager relies on the records to the aggregator model if the model is obsolete it commands it would first start training a new model.
8. The aggregator model trains the models it currently contains and asks for the models' verdict, or just gets the models' verdicts if their train date is ok.

9. The model is serialized and saved to disk for persistent storage.
10. The model manager commands the database connector to persist the enriched records. The enriched records now contain the current classifiers' verdict as well.
11. The database connector manages the interaction with the Database and persists the enriched records in the Database.

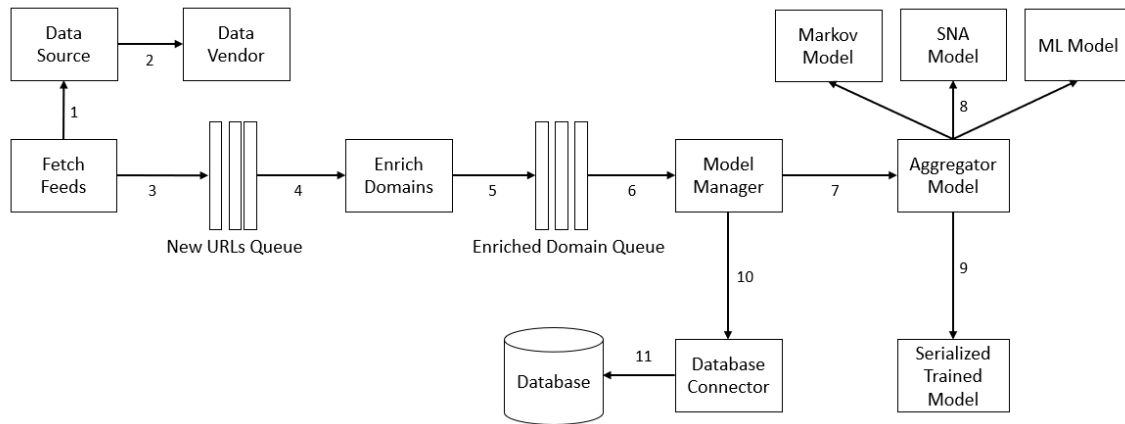


Figure 7 - Operational system flow

3.3 Domain Classifiers

This section covers the algorithm used in the domain classifier model implementation.

3.3.1 Social Network Analysis over Domain-IP Relationships

The SNA classifier was inspired by the paper “a Topology Based Flow Model for Computing Domain Reputation” [1]. The paper relies on the Domain-IP relationships which were proven to be useful for calculating domain reputation scores by the Notos system [2]. However, instead of using a machine learning classifier, it uses an interesting approach based on social network analysis (SNA) algorithm, commonly used for computing trust in social networks and virtual communities. The goal of the flow algorithm is to assign domains with reputation scores given an initial list of domains with a known reputation, good or bad.

3.3.1.1 Train – Graph Construction

In [1] the flow algorithm training contains four steps, as shown in figure 8:

- Graph construction - Create the topology graph, assign weights and represent as an adjacency matrix
- Vector - Create the initial vector used for propagation
- Iterative reputation flow - Use the vector and the matrix as input to the flow algorithm
- Final - output final reputation scores

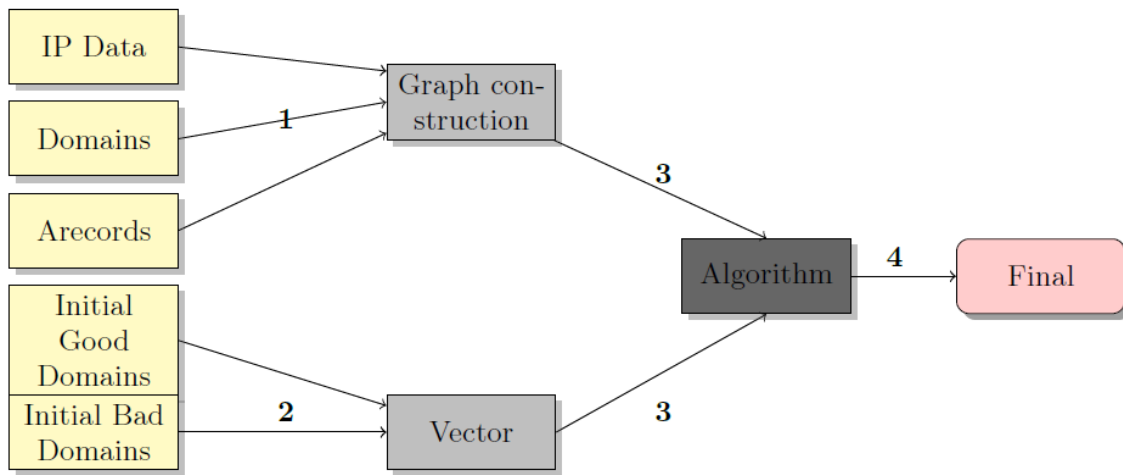


Figure 8 - A topology-based flow model for computing domain reputation [1] architecture

Unlike the original paper, I found it to be more useful to work on a graph data structure than on an adjacency matrix. That is because an adjacency matrix is less intuitive and less efficient from a performance point-of-view. Another difference from the original paper is the entities I used on the graph. I leverage all the enriched data described in the “Data Enrichment” section. Adding the “NS” DNS records and ASN data, while the original paper used only “A” records. This results with differences in the meaning of edges as is explained next.

Figure 9 shows the functions involved in the training phase which construct the graph. “train” is the model interface function. For every record in the given input dataset, it calls “_append_row_to_graph” which append a single enriched domain record to the graph. Notice that the record contains the ground truth label as well. Other than the label domain, 0 for benign and 1 for malicious, all the of the other nodes get the initial

value of 0.5. Lines 7-15 are adding the nodes to the graph, lines 17-25 are adding the edges between the nodes that were previously added. This means that edges represent different types of connections, not only domain-IP connections such as IP-AS domain-Subnet, AS name-AS number and more as listed in Figure 9. INITIAL_VALUE is the default value for the non-labeled nodes. In the experiments described in section 4, the INITIAL_VALUE was set to 0.5.

```

1  def _append_row_to_graph(self, row, G):
2      if 'label' in row:
3          G.add_node(row['domain'], start=row['label'], current=row['label'])
4          G.add_node(row['domain_ip'], start=row['label'], current=row['label'])
5      else:
6          G.add_node(row['domain'], start=INITIAL_VALUE, current=INITIAL_VALUE)
7          G.add_node(row['domain_ip'], start=INITIAL_VALUE, current=INITIAL_VALUE)
8
9      G.add_node(row['as_subnet'], start=INITIAL_VALUE, current=INITIAL_VALUE)
10     G.add_node(row['as_number'], start=INITIAL_VALUE, current=INITIAL_VALUE)
11     G.add_node(row['as_name'], start=INITIAL_VALUE, current=INITIAL_VALUE)
12     G.add_node(row['ns_base_domain'], start=INITIAL_VALUE, current=INITIAL_VALUE)
13     G.add_node(row['ns_as_subnet'], start=INITIAL_VALUE, current=INITIAL_VALUE)
14     G.add_node(row['ns_as_number'], start=INITIAL_VALUE, current=INITIAL_VALUE)
15     G.add_node(row['ns_as_name'], start=INITIAL_VALUE, current=INITIAL_VALUE)
16
17     if row['base_domain'] != row['domain']:
18         G.add_node(row['base_domain'], start=INITIAL_VALUE, current=INITIAL_VALUE)
19         G.add_edge(row['base_domain'], row['domain'])
20
21     G.add_edge(row['domain'], row['domain_ip'])
22     G.add_edge(row['domain_ip'], row['as_subnet'])
23     G.add_edge(row['as_subnet'], row['as_number'])
24     G.add_edge(row['as_number'], row['as_name'])
25
26     G.add_edge(row['base_domain'], row['ns_base_domain'])
27     G.add_edge(row['ns_base_domain'], row['ns_as_subnet'])
28     G.add_edge(row['ns_as_subnet'], row['ns_as_number'])
29     G.add_edge(row['ns_as_number'], row['ns_as_name'])

```

Figure 9 - Append enriched domain record to the graph function

Figure 10 shows the visualization of a trained graph. When we add the unlabeled enriched node “*edition.cnn.com*”, additional nodes and edges are created: green nodes are labeled as benign, red are labeled as malicious and brown nodes are unlabeled. As expected most of the neighbors of “*edition.cnn.com*” are benign or unlabeled. The only red node in the graph is a subdomain of a freemium hosting service. Freemium hosting

is a service that offers free basic web services deployment and a paid fully-suite package. In this case, codeanywhere.com is a freemium service that was abused for malicious purposes.

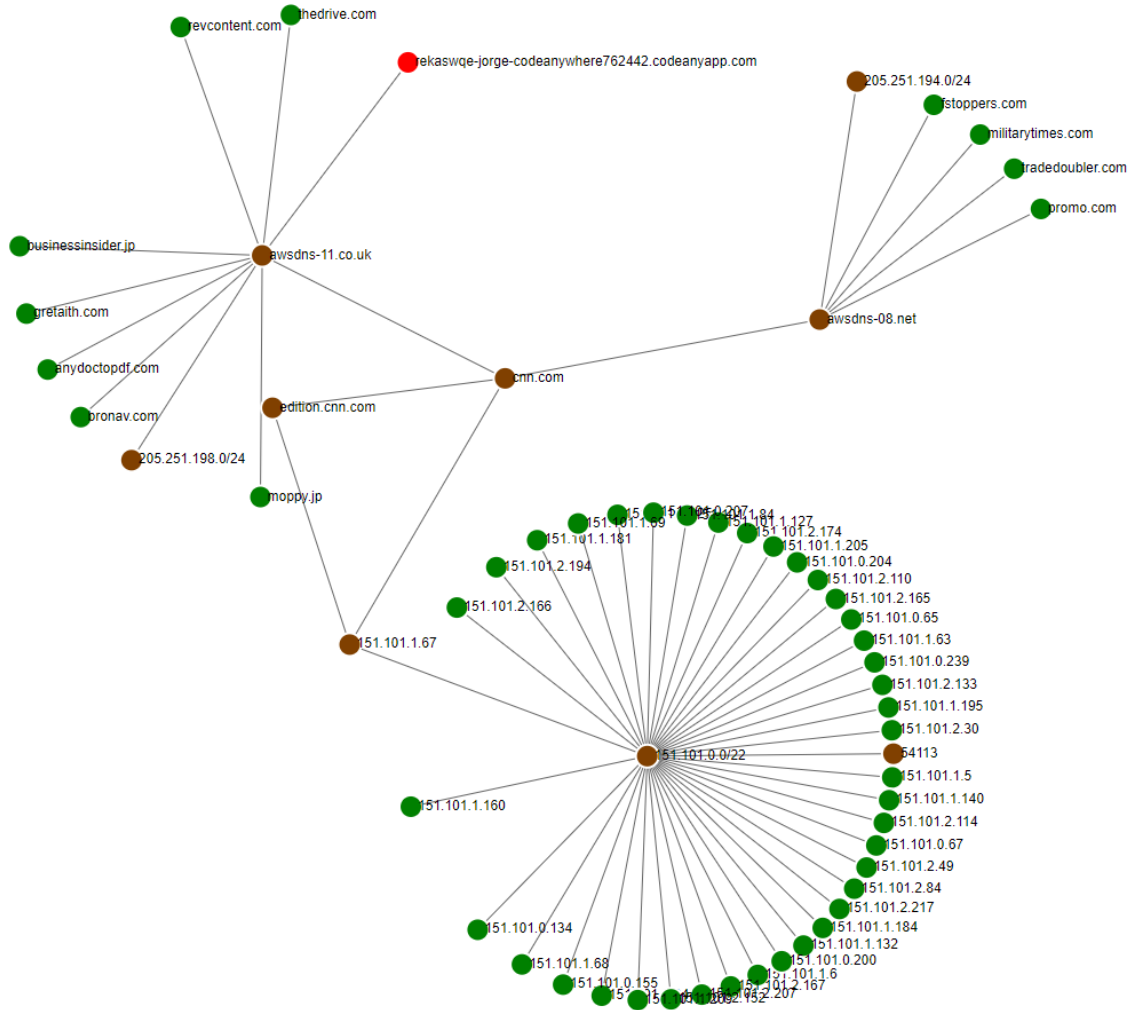


Figure 10 - Ego graph with radius 3 of edition.cnn.com

3.3.1.2 Predict – Graph Iterations

Figure 10 shows the functions involved in the predict phase. “predict” is the model interface function. In line 2 It creates a copy of the previously constructed graph. In Line 3 it appends the given records (X) to the graph copy. Line 4 class calls “_stable_graph” that is defined in figure 10 as well. “_stable_graph” is the phase where the reputation of each node cascades to its neighbors. On the worse case, each iteration may cause $O(|G.nodes|)$ updates on the graph. That is why its wise to limit the amount of the

iterations. My analysis shows that 5 iterations are enough, [1] reached a similar conclusion on their implementation. In each iteration every node score is set to be the average between its current score and the average score of its neighbors, see the “_update_node” function in figure 11.

```
1  def predict(self, X):
2      H = self._G.copy()
3      X.apply(self._append_row_to_graph, args=(H,), axis=1)
4      self._stable_graph(H, iterations=5)
5
6  def _stable_graph(self, graph, iterations=10):
7      for _ in range(iterations):
8          self._graph_iteration(graph)
9
10 def _graph_iteration(self, graph):
11     for node in graph.nodes():
12         self._update_node(graph, node)
13
14 def _update_node(self, graph, node):
15     if graph.degree(node) > 0:
16         neighbors_current_score = 0
17         for neighbor in graph.neighbors(node):
18             neighbors_current_score += graph.nodes[neighbor]['current']
19         neighbors_current_avg = \
20             neighbors_current_score / graph.degree(node)
21         graph.nodes[node]['current'] = \
22             0.5 * graph.nodes[node]['current'] + 0.5 * neighbors_current_avg
```

Figure 11 - SNA model prediction process

3.3.2 Machine Learning Blacklisting at Time-of-Registration

WHOIS record is the data describing the registration of the domain such as registration date, last modified date, expiration date, registrant contact information, registrar contact information and nameserver domains. Once the domain has been registered, the relevant registry is the owner of the WHOIS database record. WHOIS-based reputation approach advantage is that it could be the first line of defense in detecting new malicious domains and that it enables following threat actors reusing domain registration information. Its drawback is that WHOIS information is often anonymized or only partly available as each registry information is not standard for WHOIS record

completeness. In this section, I describe an implementation based on the domain reputation System called PREDATOR, described in the paper “Proactive Recognition and Elimination of Domain Abuse at Time-Of-Registration” [3].

PREDATOR is a system aimed to achieve early detection of malicious domains by using only WHOIS records as input. The paper contains a description of the feature engineering process resulting in 22 features types. The features can help distinguish abusive domain registration behavior characteristics from legitimate registration behavior characteristics. These features are fed into a state-of-the-art supervised learning algorithm.

3.3.2.1 Train – Nameserver Features

PREDATOR system architecture shown in figure 12 is very similar to the one used in this project which includes a training mode and an operation mode.

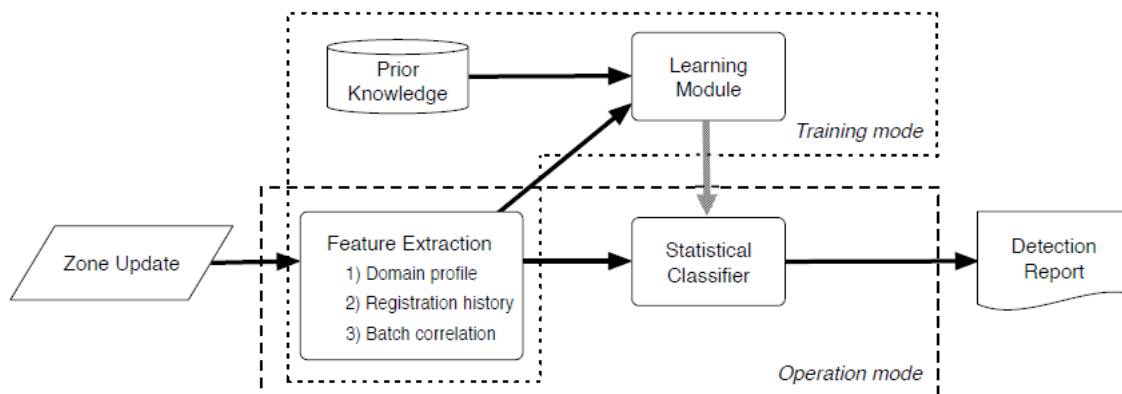


Figure 12 - A high-level overview of PREDATOR [3] architecture

Table 5 shows the 22 types of features used by PREDATOR. These features are divided into three groups: domain profile features, registration history features, and batch correlation features. Unfortunately, the data on registration history features and batch correlation features is not publicly available. Therefore, this project focus on domain profile features only.

<i>Category</i>	<i>Feature</i>	<i>Type</i>
Domain profile	Registrar	Categ.
	Authoritative nameservers	Categ.
	IP addresses of nameservers	Categ.
	ASes of nameserver IP addresses	Categ.
	Daily hour of registration	Categ.
	Week day of registration	Categ.
	Length of registration period	Ord.
	Trigrams in domain name	Categ.
	Ratio of the longest English word	Cont.
	Containing digits	Categ.
	Containing “_”	Categ.
Registration history	Name length	Ord.
	Edit distances to known-bad domains	Cont.
	Life cycle	Categ.
	Dormancy period for re-registration	Ord.
Batch correlation	Previous registrar	Categ.
	Re-registration from same registrar	Categ.
	Probability of batch size	Cont.
	Brand-new proportion	Cont.
	Drop-catch proportion	Cont.
	Retread proportion	Cont.
	Name cohesiveness	Cont.

Table 5: Summary of PREDATOR [3] features, each feature is categorical, continuous or ordinal.

Table 6 shows the PREDATOR feature importance. From the table, we can see that the focus on domain profile features is reasonable since the top 6 features out of the 22 and the top 7 out of the top 8 are domain profile features. The features I selected to implement in the project are:

- Authoritative nameservers (ranked #1), to increase the detection rate the base domain of the nameserver was used the feature.
- IP addresses of nameservers (ranked #3)
- ASes of nameserver IP addresses (ranked #5)

The project doesn't contain the following features that were presented in PREDATOR:

- Trigrams in the domain name (ranked #2) cause a massive increase in the number of features and led the classifier to be slow, heavy and tend for overfitting.

- Registrar (ranked #4), this feature can be extracted only from a premium paid feed. which conflicts with one of the project's secondary goals to the classifiers to be based on free and open repositories only.
- Daily hour of registration & Weekday of registration (ranked #6 and #8)
This data simply not publicly available in any form.

<i>Rank</i>	<i>Category</i>	<i>Feature</i>	<i>Score ratio</i>
1	D	Authoritative nameservers	100.00%
2	D	Trigrams in domain name	64.88%
3	D	IP addresses of nameservers	62.98%
4	D	Registrar	61.28%
5	D	ASes of nameserver IP addresses	30.80%
6	D	Daily hour of registration	30.30%
7	B	Name cohesiveness	28.98%
8	D	Weekday of registration	22.58%
9	R	Dormancy period for re-registration	20.58%
10	R	Re-registration from same registrar	19.50%
11	R	Life cycle	18.55%
12	D	Edit distances to known-bad domains	17.72%
13	R	Previous registrar	16.50%
14	B	Brand-new proportion	14.60%
15	B	Retread proportion	13.71%
16	B	Drop-catch proportion	12.90%
17	D	Containing digits	11.25%
18	D	Name length	10.71%
19	D	Ratio of the longest English word	9.60%
20	B	Probability of batch size	8.66%
21	D	Containing “_”	8.02%
22	D	Length of registration period	3.34%

Table 6 - Ranking of feature importance in PREDATOR [3] (D for domain profile category, R for registration history category, and B for batch correlation category).

The selected features are categorical, therefore they are translated into binary features since binary features are more common for training Machine Learning models. Table 7 shows an example of categorical features the model decodes into binary features.

domain	ns_base_domain	ns_as_subnet	ns_as_name	label
paypal.com.user-login.secure-id.ref939a.com	dendrite.network	45.9.148.0/24	NICEIT, NL	1
paypalaccounttologinaccountssummarmay.com	ispvds.com	94.250.248.0/23	THEFIRST-AS, RU	1
paypal-id-signin-customer-center-customer-locale-g-en.c	freenom.com	104.155.0.0/19	GOOGLE - Google LLC, US	1
paypal.com.au-dispute50043.gajsiddhiglobal.com	speedhost.in	208.91.198.0/23	PUBLIC-DOMAIN-REGISTRY - PDR, US	1
paypal-limitato-conferma.kozow.com	dynu.com	45.79.208.0/20	LINODE-AP Linode, LLC, US	1
paypal.co.uk.3uea.icu	dnspod.com	119.28.48.0/23	TENCENT-NET-AP-CN Tencent Building, Kejizhongyi Avenue, CN	1
paypal.co.uk.v15m.icu	dnspod.com	180.160.0.0/13	CHINANET-SH-AP China Telecom (Group), CN	1
paypal.de-center.buzz	cloudflare.com	173.245.59.0/24	CLOUDFLARENET - Cloudflare, Inc., US	1
paypal.co.uk.dii7.icu	dnspod.com	59.36.112.0/20	CHINANET-IDC-GD China Telecom (Group), CN	1
paypal-webnative.surge.sh	iwantmyname.net	83.169.54.0/23	GODADDY, DE	1
checkout.paypal.com	ultradns.net	156.154.65.0/24	ULTRADNS - NeuStar, Inc., US	0
c6.paypal.com.edgekey.net	akamai.net	95.100.173.0/24	AKAMAI-ASN2, US	0
api-m-edge.glb.paypal.com	dyndns.net	204.13.250.0/24	DYNDNS - Oracle Corporation, US	0
svcs.paypal.com	ultradns.net	156.154.65.0/24	ULTRADNS - NeuStar, Inc., US	0
paypal.me	dyndns.net	204.13.250.0/24	DYNDNS - Oracle Corporation, US	0
paypal-deutschland.de	dyndns.net	208.78.70.0/24	DYNDNS - Oracle Corporation, US	0
paypal.com.au	ultradns.net	156.154.65.0/24	ULTRADNS - NeuStar, Inc., US	0
paypal-business.co.uk	dyndns.net	208.78.70.0/24	DYNDNS - Oracle Corporation, US	0

Table 7 - Records from the dataset for building an example model for PayPal phishing detection

For the 18 records shown in table 7, there is a limit of $18 * 3 = 54$ decode features. Table 8 continues the example in table 7. It shows the decoding result which ended with 11 ns_base_domain feature, 14 ns_as_subnet features, and 13 ns_as_name features. Total of 38 features. The greater the dataset, the lower is the ratio between the maximal amount of decoded features and the resulted amount. That is due to the repeatedness of the features.

ns_base_domain	ns_as_subnet	ns_as_name
ns_base_domain_akamai.net	ns_as_subnet_104.155.0.0/19	ns_as_name_AKAMAI-ASN2, US
ns_base_domain_cloudflare.com	ns_as_subnet_119.28.48.0/23	ns_as_name_CHINANET-IDC-GD China Telecom (Group), CN
ns_base_domain_dendrite.network	ns_as_subnet_156.154.65.0/24	ns_as_name_CHINANET-SH-AP China Telecom (Group), CN
ns_base_domain_dnspod.com	ns_as_subnet_173.245.59.0/24	ns_as_name_CLOUDFLARENET - Cloudflare, Inc., US
ns_base_domain_dynect.net	ns_as_subnet_180.160.0.0/13	ns_as_name_DYNDNS - Oracle Corporation, US
ns_base_domain_dynu.com	ns_as_subnet_204.13.250.0/24	ns_as_name_GODADDY, DE
ns_base_domain_freenom.com	ns_as_subnet_208.78.70.0/24	ns_as_name_GOOGLE - Google LLC, US
ns_base_domain_ispvds.com	ns_as_subnet_208.91.198.0/23	ns_as_name_LINODE-AP Linode, LLC, US
ns_base_domain_iwantmyname.net	ns_as_subnet_45.79.208.0/20	ns_as_name_NICEIT, NL
ns_base_domain_speedhost.in	ns_as_subnet_45.9.148.0/24	ns_as_name_PUBLIC-DOMAIN-REGISTRY - PDR, US
ns_base_domain_ultradns.net	ns_as_subnet_59.36.112.0/20	ns_as_name_TENCENT-NET-AP-CN Tencent Building, Kejizhongyi Avenue, CN
	ns_as_subnet_83.169.54.0/23	ns_as_name_THEFIRST-AS, RU
	ns_as_subnet_94.250.248.0/23	ns_as_name_ULTRADNS - NeuStar, Inc., US
	ns_as_subnet_95.100.173.0/24	

Table 8 - Decoded features for the records of table 7.

For the phishing domain *paypalaccounttologinaccountssummarmay.com* the feature vector would be “ns_base_domain_ ispvds.com”, “ns_as_subnet_94.250.248.0/23”, and “ns_as_name_THEFIRST-AS, RU” set to 1. The other features would be set to 0.

3.3.2.2 Predict – Scikit-Learn Complaint

The MLModel class is compatible with the Scikit-Learn interface. In the project experiments section, I'll elaborate on the tested models and the results.

3.3.3 Predictive Blacklisting

Predictive blacklisting approach leverage existing knowledge of malicious domains to predict malicious domain names that are likely to be used for malicious purposes. The approach advantage that it could be the first line of defense in detecting new malicious domains. It is based on the empirical fact that threat actors reusing domain name template strings with minor edits. Its drawbacks are that it counts on threat actors' lack of imagination in picking phishing domain names and that most of its output is redundant since most of the domains it generates are never in use. The paper Proactive discovery of phishing related domain names [4] describes such a system. The paper describes a system that generates a blacklist of domains by using a Markov chain model and relevant lexical features extracted from a semantic splitter. Domain-specific knowledge added from semantic tools.

3.3.3.1 Train – Markov Chain

The proactive malicious domain name discovery training contains six steps, as shown in figure 13:

1. Information gathering – collect top-level domain (TLD) from the public suffix list and malicious domains as input for the proactive model
2. Name decomposition – break down the main domain name and TLD
3. Word splitter – break down main domain name into words
4. Model – run the statistical analysis and predict potential malicious domains list
5. Domain checker – filter benign domains before adding to a blacklist
6. Blacklist – publish a blacklist of potential malicious domains

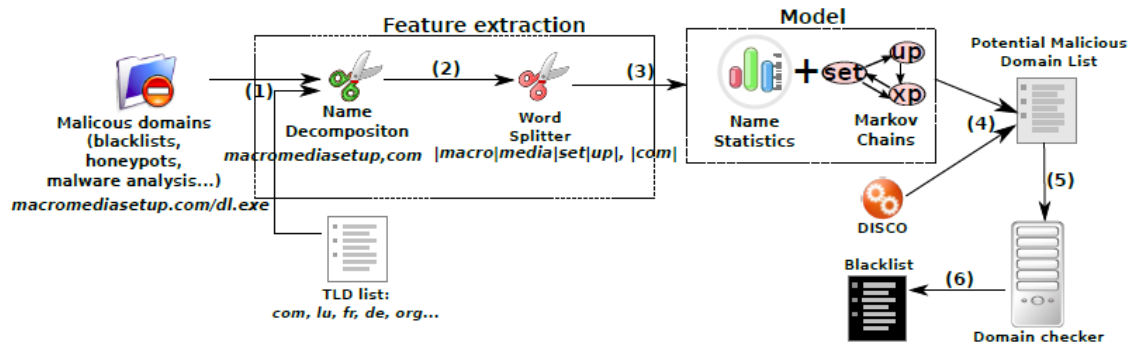


Figure 13: Proactive Malicious Domain Name Discovery System [4] architecture

In this project, the name decomposition was done with Compound-Word-Splitter python package. After the name decomposition phase words statistics are gathered. Figure 14 shows an example of words statistics gathering for the word “free” and the word “pay”. In the case of the word “free”, the next transition in the Markov chain would be any one of the words in the “transitions” counter. Since all the following words have the same amount of apparencies following the word “free”, they would get the same probability for the next phase: $1/11 = 0.090909$. In the case of the word “pay,” the next transition in the Markov chain would be any one of the words in the “transitions” counter. Since all the following words but the word “problems” have the same amount of apparencies following the word “free”, they would get the same probability for the next phase: $1/13 = 0.076923$ and the word “problems” which appeared twice, it’s probability would be $2/13 = 0.153846$.

1	{	1	{
2	'appearance': 15,	2	'appearance': 17,
3	'index': Counter({	3	'index': Counter({
4	0: 8,	4	0: 9,
5	1: 3,	5	1: 5,
6	2: 3,	6	3: 3
7	3: 1	7	}),
8	}),	8	'sentence_length': Counter({
9	'sentence_length': Counter({	9	3: 6,
10	2: 4,	10	4: 5,
11	3: 7,	11	2: 5,
12	5: 1,	12	5: 1
13	4: 3	13	}),
14	}),	14	'transitions': Counter({
15	'transitions': Counter({	15	'pay': 0,
16	'free': 0,	16	'problems': 2,
17	'liker': 1,	17	'la': 1,
18	'o': 1,	18	'x': 1,
19	'you': 1,	19	'certain': 1,
20	'get': 1,	20	'v': 1,
21	'click': 1,	21	'pack': 1,
22	'gift': 1,	22	'io': 1,
23	'l': 1,	23	'you': 1,
24	'game': 1,	24	'm': 1,
25	'ia': 1,	25	'only': 1,
26	'host': 1,	26	'bank': 1,
27	'movies': 1	27	'7158': 1
28	})	28	})
29	}	29	}

Figure 14 – on the left word statistics of the word “free”, and on the right word statistics for the word “pay”

The decision to end the domain name, i.e. not to continue with another transition, is made using the “sentence_length” field in the “word statistics” data structure as shown in figure 14, lines 9-14 left and lines 8-13 right. The stop criteria is based on the sentence words length statistics of the last word in the generated domain name. The stop criteria is shown in line 12 in figure 15.


```

1   def _create_random_domain_name(self, model, word_statistics, initial_state):
2       domain_name = None
3       word_list = [initial_state]
4       current_state = initial_state
5       while len(word_statistics[current_state]['transitions']) > 0:
6           current_state = model.next_state(current_state)
7           word_list.append(current_state)
8           prob_dict = self._convert_counter_to_probabilities(
9               word_statistics[current_state]['sentence_length'])
10          current_word_sentence_length = np.random.choice(
11              list(prob_dict.keys()), p=list(prob_dict.values()))
12          if current_word_sentence_length <= len(word_list):
13              break
14          if len(word_list) > 1:
15              domain_name = ''.join(word_list)
16          return domain_name

```

Figure 15 - Markov model domain names generator

Figure 16 shows the algorithm that creates the blacklist of predicted malicious domain names. For every word in the “word statistics” data structure described in figure 13, the algorithm generates up to 100 predicted malicious domain names. If the algorithm spots that the generated domains repeat more than 10 times, it continues to the next word.

```

1   self.domain_name_set = set()
2   for word in self._states_set:
3       init_set_len = len(self.domain_name_set)
4       for i in range(100):
5           predict_domain_name = self._create_random_domain_name(
6               self._markov_chain, self._word_statistics, word)
7           if predict_domain_name is not None \
8               and predict_domain_name not in domain_name_blacklist \
9               and predict_domain_name not in self.domain_name_set \
10              and len(predict_domain_name) > 5:
11               self.domain_name_set.add(predict_domain_name)
12           elif predict_domain_name is None \
13               or len(self.domain_name_set) + 10 < init_set_len + i:
14               break

```

Figure 16 - Creating a blacklist of predicted malicious domains

3.3.3.2 Predict

The predict function is trivial, it just checks if the domain name appears in the predicted domain name set that was created using the algorithm shown in figure 16.

4. Experiment

This section covers the process of the experiments: data cleaning, threshold selection, and classifier result evaluation.

4.1 Data Cleaning

In the early stages of the experiment, an anomaly popped up. Many phishing domains were hosted as a subdomain of popular hosting websites such as 000webhostapp.com, azurewebsites.net, duckdns.org, no-ip.com, no-ip.org, wixsite.com. The mentioned domains offer a freemium hosting service. Threat actor takes advantage of these freemium services for their malicious purpose. In order to avoid causing confusion to the classifiers, malicious domains hosted on the mentioned hosting providers were removed from the train and test set. It reduced 32% of the dataset. That is not a great loss since these domains could not be analyzed by the classifier developed in this project anyway since the top domain is always benign.

In the middle of the experiments, I notice a sharp increase in the detection rate of the ML models and a decline in the SNA model. The reason for that was many domain records had not IP, nameserver, and ASN data. It was caused due to a networking failure that I didn't handle properly. I fix the code and removed the empty domain record from the dataset.

4.2 Data Separation

To decide which threshold every classifier should have I've visualized all the classifiers' verdicts into the charts seen in figure 17.

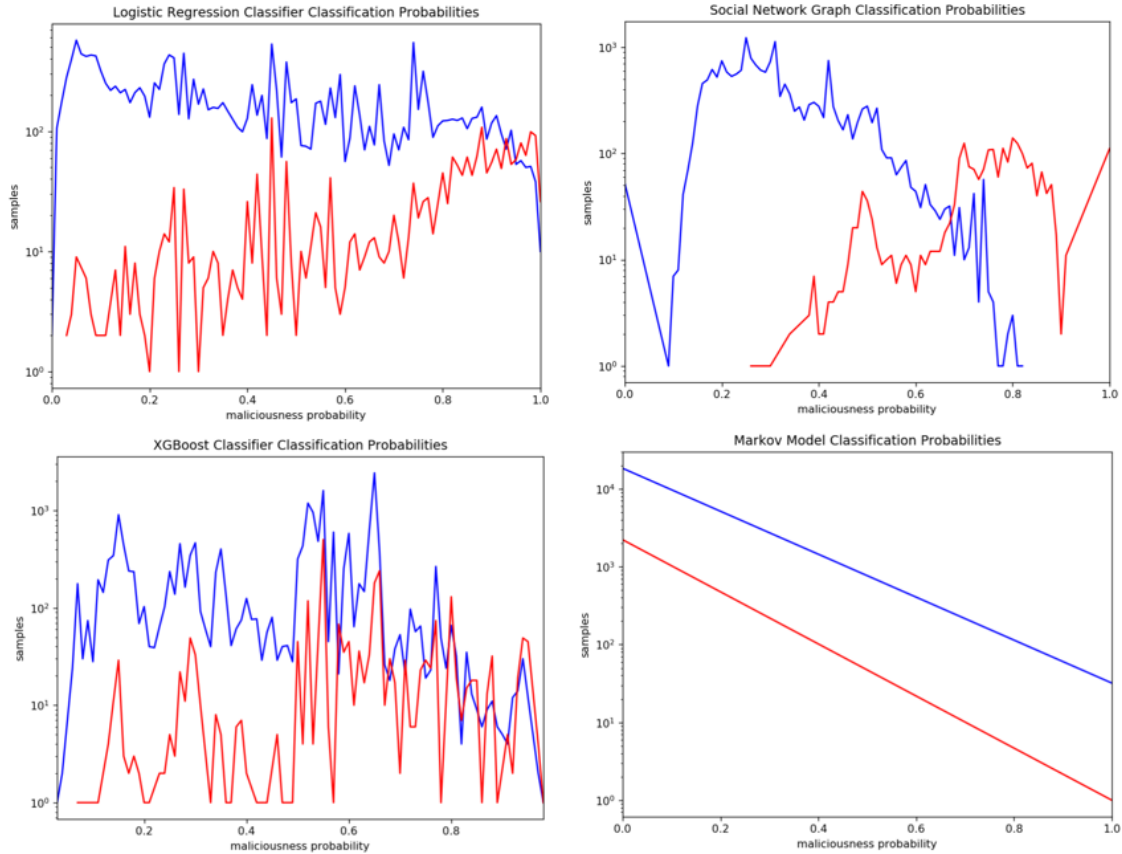


Figure 17 - Classification data separation charts. Blue represents benign sample probabilities, and red are malicious sample probabilities.

The classifiers' threshold selection is an important part of the experiment. The optimal threshold is the one that conducts a perfect separation between the classes. In our case the separation between benign and malicious domains. Since in real life the optimal threshold is not perfect, we'll select a threshold that maximizes true positives and at cost of minimal false positives. In figure 17 we can see the places the red line is high then the blue line. For the Markov model, the threshold is a Boolean threshold, but unfortunately in the experiment, it had more false positives in any threshold. The selected thresholds are listed in table 9.

Model	Threshold
SNA	0.67
XGBoost	0.93
Logistic Regression	0.95
Markov	1

Table 9 - The selected threshold for the classifiers

4.3 Evaluation

After the classifiers' decision threshold was set its possible to translate the classifiers' probabilities results into verdicts. The evaluation was made on data collected between 17-Dec-2019 and 23-Dec-2019. In that time period, 20,640 labeled domain samples were collected. 18,148 labeled as benign and 2,222 labeled as malicious. Table 10 shows a clear advantage of the SNA classifier which produces a detection rate of 83.89% at the price of 1.09% false positive rate.

	Logistic Regression			
	Raw		Normalized	
	Benign	Malicious	Benign	Malicious
Benign	18186	232	98.74%	1.26%
Malicious	1833	389	82.49%	17.51%
	XGBoost			
	Raw		Normalized	
	Benign	Malicious	Benign	Malicious
Benign	18370	48	99.74%	0.26%
Malicious	2111	111	95.00%	5.00%
	SNA			
	Raw		Normalized	
	Benign	Malicious	Benign	Malicious
Benign	18218	200	98.91%	1.09%
Malicious	358	1864	16.11%	83.89%

Table 10 - Confusion matrixes of the classifiers' evolution

The ROC curve shown in figure 18, confirms the SNA model out-perform the other classifier on any given threshold. You can see its line always above the Logistic Regression and XGBoost classifier. It's also interesting to see that the simplistic Logistic

Regression algorithm out-perform the state-of-the-art machine learning algorithm XGBoost.

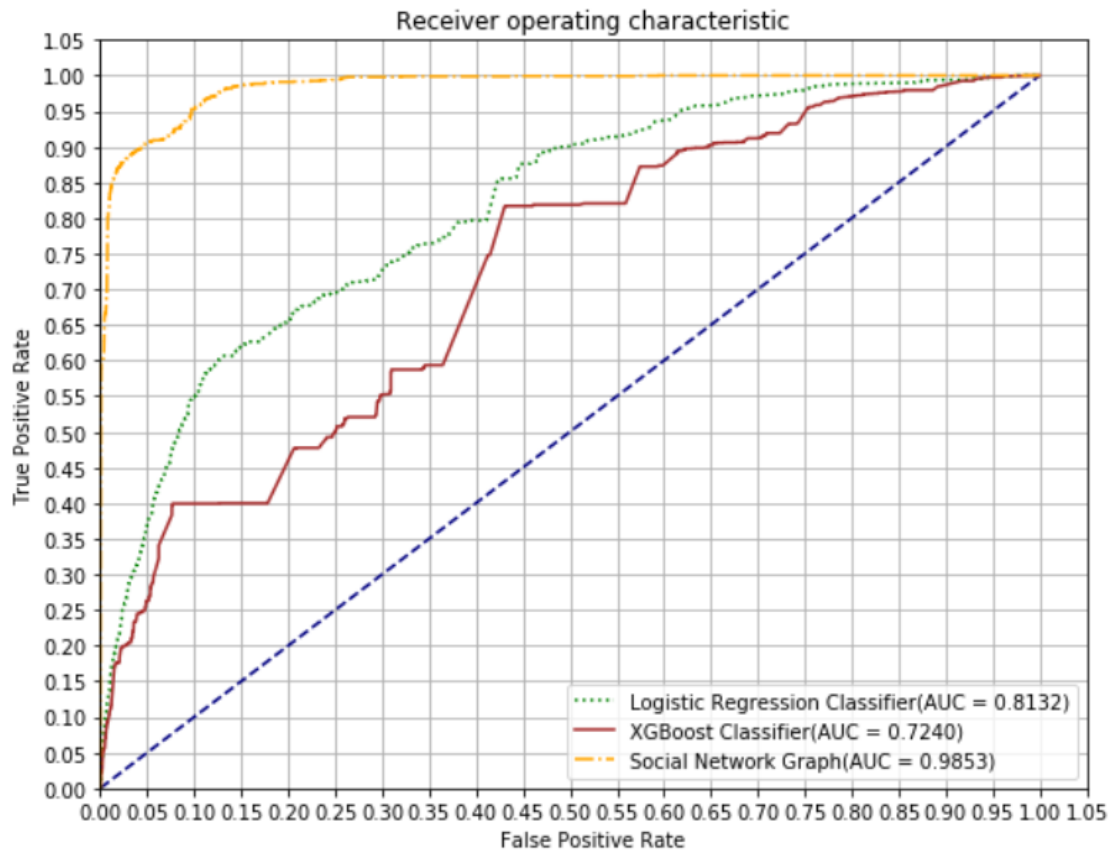


Figure 18 - ROC Curve using the threshold shown in table 9

The PREDATOR [3] system baseline its evaluation of on a given FPR of 0.35%. For the results to be comparable with each other, I did the same. PREDATOR results are shown in Table 11, the project results are shown in Table 12. The results show that the SNA model reaches a similar detection rate to PREDATOR. That is without optimization tuning of the training and testing window size as done in PREDATOR paper. In the experiment the SNA model was rebuilt every 2 hours, that is possible since the graph construction takes less than a minute. PREDATOR paper doesn't specify how much time it takes to train the model, but I guess it's much more than a minute.

Testing window / Training window	Testing window		
	7 days	35 days	56 days
35 days	70.00%	68.29%	66.81%
21 days	67.10%	64.96%	60.56%
14 days	64.13%	60.51%	58.22%

Table 11 - PREDATOR [3] detection rates under a 0.35% false positive rate

Model	TPR
SNA	60.71%
XGBoost	5.40%
Logistic Regression	7.56%
Markov	0.00%

Table 12 - Classifier detection rate under a 0.35% false positive rate

Figure 19 shows the ROC curve of PREDATOR. When comparing to the ROC of the SNA model shown in figure 18, it's clearly shown that the SNA ROC curve compensates better for a more tolerance FPR. For example, when considering an FPR of 1% the SNA model obtains 82.81% TPR, when according to figure 19 PREDATOR obtains less than 80%.

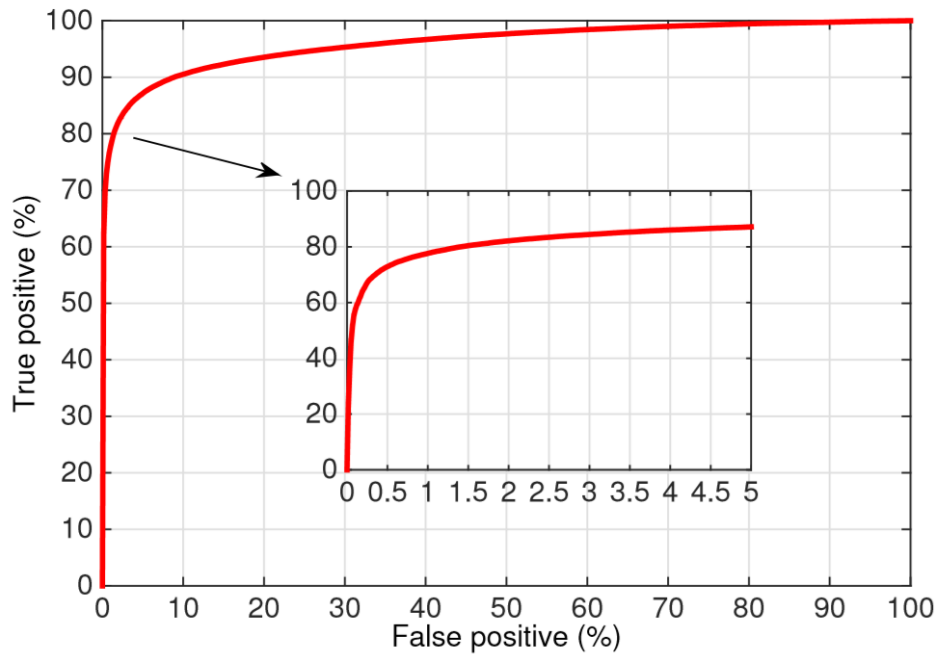


Figure 19 - ROC of PREDATOR [3] The inlay figure shows the ROC curve under the range of 0–5% false positives

5. Project Configurations and Operation

This section explains how to configure the project environment, data sources, technologies and how to operate the project.

5.1 Setup

This section explains the project preliminary requirements before running it.

5.1.1 Software Components

The project requires Python 3.6 or newer to install it download go to

<https://www.python.org/downloads/>

Once Python is installed on the machine, you'll need to install the external Python package that is in use in this project. To install the packages all you need is to run the following command on the project directory shell: `pip install -r requirements.txt`

The project's persistent storage is PostgreSQL, however, it can be easily ported to another database technology. To install PostgreSQL download it from

<https://www.postgresql.org/download/> and following the installation instruction.

The project global cache and messaging queue mechanized is Redis. To install Redis download it from <https://redis.io/download> and following the installation instruction.

5.1.2 Data Source Configurations

As mentioned in the data collection section, to harvest malicious URLs from PhishTank you'll need an API key. You can get it for free on the following registration URL:

<https://www.phishtank.com/register.php>. The API Key should be stored in the operating system environment variable `PHISHTANK_APIKEY`.

The ASN enrichment is done with the `pyasn` python package. It requires downloading the freely available MRT/RIB BGP archives. The download and installation process is explained on the `pyasn` GitHub page: <https://github.com/hadiasghari/pyasn>. The ASN database file path should be stored on the operating system environment variable `ASN_DB_PATH`. The ASN database is changing on a daily basis, thanks to `pyasn` it's easy to update it.

5.1.3 Logging

Logging in the project is made with the Python built-in logging package. The desired log directory location should be stored on the operating system environment variable LOG_DIR_PATH.

5.2 Running the Backend

The project has three Python files with the main function: fetch-feeds.py, enrich_domain.py, and model_managaer.py.

fetch-feeds.py is the python script that fetches the domain and URL feed from the data sources mentioned in the data collection section (3.2.1). After fetching the feed, it transmits the domains to the new URLs channel. In figure 20 you can see its manual.

```
usage: fetch_feeds.py [-h] [--infinity] [--sleep SLEEP]
                    [--data-source DATA_SOURCE] [--redis-host REDIS_HOST]
                    [--redis-port REDIS_PORT] [--redis-db REDIS_DB]
                    [--limit LIMIT] [--debug-level DEBUG_LEVEL]

optional arguments:
  -h, --help            show this help message and exit
  --infinity            infinte run
  --sleep SLEEP         sleep seconds, relevant only on infinity mode
  --data-source DATA_SOURCE
                        data source to fetch
  --redis-host REDIS_HOST
                        redis hostname
  --redis-port REDIS_PORT
                        redis port
  --redis-db REDIS_DB   redis db index
  --limit LIMIT         publish limit for fetched URLs
  --debug-level DEBUG_LEVEL
                        logging debug level
```

Figure 20 - fetch-domains.py manual

enrich_domain.py is the python script that listens to new URLs channel and enriches the domains which IP, nameserver and ASN data. In figure 21 you can see its manual.


```
usage: enrich_domain.py [-h] [--postgresql-host POSTGRESQL_HOST]
                        [--postgresql-port POSTGRESQL_PORT]
                        [--postgresql-username POSTGRESQL_USERNAME]
                        [--postgresql-password POSTGRESQL_PASSWORD]
                        [--debug-level DEBUG_LEVEL]

optional arguments:
  -h, --help            show this help message and exit
  --postgresql-host POSTGRESQL_HOST
                        postgresql host
  --postgresql-port POSTGRESQL_PORT
                        postgresql port
  --postgresql-username POSTGRESQL_USERNAME
                        postgresql username
  --postgresql-password POSTGRESQL_PASSWORD
                        postgresql password
  --debug-level DEBUG_LEVEL
                        logging debug level
```

Figure 21 - fetch-domains.py manual

model_manager.py is a python script that listens to enriched domains channel. It runs the classifiers on the enriched dataset and commits the domain record with the classifier results to the database. In figure 22 you can see the script manual.

```
usage: model_manager.py [-h] [--train] [--listen] [--logger LOGGER]
                       [--pkl-path PKL_PATH] [--limit LIMIT]
                       [--retrain RETRAIN]
                       [--postgresql-host POSTGRESQL_HOST]
                       [--postgresql-port POSTGRESQL_PORT]
                       [--postgresql-username POSTGRESQL_USERNAME]
                       [--postgresql-password POSTGRESQL_PASSWORD]
                       [--debug-level DEBUG_LEVEL]

optional arguments:
  -h, --help            show this help message and exit
  --train               train new models
  --listen              listen for new enriched domains
  --logger LOGGER       logger name
  --pkl-path PKL_PATH   path to pickle to save/load the model file
  --limit LIMIT         limit records per classification
  --retrain RETRAIN     model retraining every X hours, 0 for no retraining
  --postgresql-host POSTGRESQL_HOST
                        postgresql host
  --postgresql-port POSTGRESQL_PORT
                        postgresql port
  --postgresql-username POSTGRESQL_USERNAME
                        postgresql username
  --postgresql-password POSTGRESQL_PASSWORD
                        postgresql password
  --debug-level DEBUG_LEVEL
                        logging debug level
```

Figure 22 - model_manager.py manual

5.3 Running the Frontend

The project frontend is written in Python as well. The backend of the frontend is on the file `app.py` and its frontend is on `templates` directory. figure 23 shows `app.py` manual.

```
usage: app.py [-h] [--flask-host FLASK_HOST] [--flask-port FLASK_PORT]
              [--logger LOGGER] [--pkl-path PKL_PATH] [--limit LIMIT]
              [--postgresql-host POSTGRESQL_HOST]
              [--postgresql-port POSTGRESQL_PORT]
              [--postgresql-username POSTGRESQL_USERNAME]
              [--postgresql-password POSTGRESQL_PASSWORD]
              [--debug-level DEBUG_LEVEL]

optional arguments:
  -h, --help            show this help message and exit
  --flask-host FLASK_HOST
                        flask host
  --flask-port FLASK_PORT
                        flask port
  --logger LOGGER        logger name
  --pkl-path PKL_PATH    path to pickle to save/load the model file
  --limit LIMIT          limit records per classification
  --postgresql-host POSTGRESQL_HOST
                        postgresql host
  --postgresql-port POSTGRESQL_PORT
                        postgresql port
  --postgresql-username POSTGRESQL_USERNAME
                        postgresql username
  --postgresql-password POSTGRESQL_PASSWORD
                        postgresql password
  --debug-level DEBUG_LEVEL
                        logging debug level
```

Figure 23 - `app.py` manual

5.4 Frontend Operation

This section explains how to operate the frontend web user interface.

5.4.1 Domain Reputation

The default screen is “Domain Reputation”. In this web page, a user can input a URL address and the URL’s domain reputation score would be calculated. Figure 24 shows the screen.

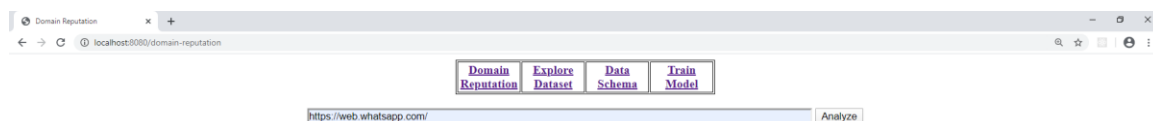


Figure 24 - "Domain Reputation" screenshot, input box filled with the URL <https://web.whatsapp.com/>.

Figure 25 shows the upper screen output for the URL <https://web.whatsapp.com/>. the output includes 3 sections: input URL and extracted domain, the results of the raw classifier, and the features extracted.

URL	https://web.whatsapp.com/			
Domain	web.whatsapp.com			

mlmodel_logisticregression	mlmodel_xgbclassifier	markovmodel	snamodel	train_date
0.190483	0.548883	0	0.388562	2019-12-30T19:46:54.053659

timestamp	2019-12-30T20:15:46.360672
base_domain	whatsapp.com
domain_name	whatsapp
domain_ip	185.60.216.53
as_number	32934
as_subnet	185.60.216.0/24
as_name	FACEBOOK - Facebook, Inc., US
nameserver	a.ns.whatsapp.net
ns_base_domain	whatsapp.net
ns_domain_ip	66.111.48.12
ns_as_number	11917
ns_as_subnet	66.111.48.0/24
ns_as_name	WHATSAPP - WhatsApp, US

Figure 25 - Domain reputation result for the URL <https://web.whatsapp.com/>

Figure 26 shows the bottom part of the screen, it presents an explanation of the SNA classifier verdict. It colors green for benign domains, brown for unknown and red for malicious domains. The edges between the nodes are constructed by the algorithm described in figure 9 from the SNA domain classifier train section (3.3.1.1).

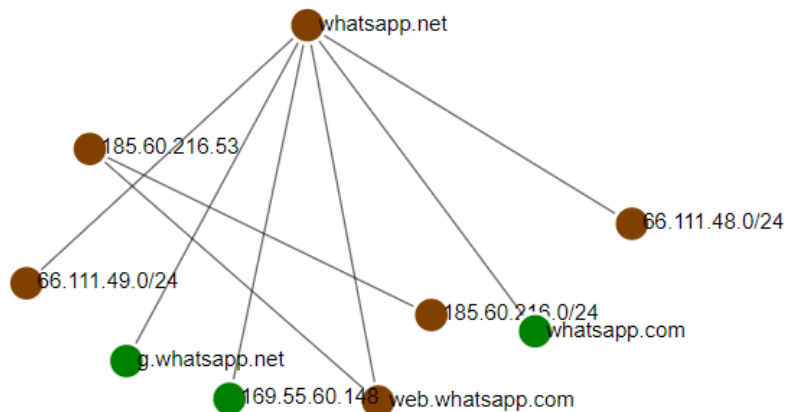
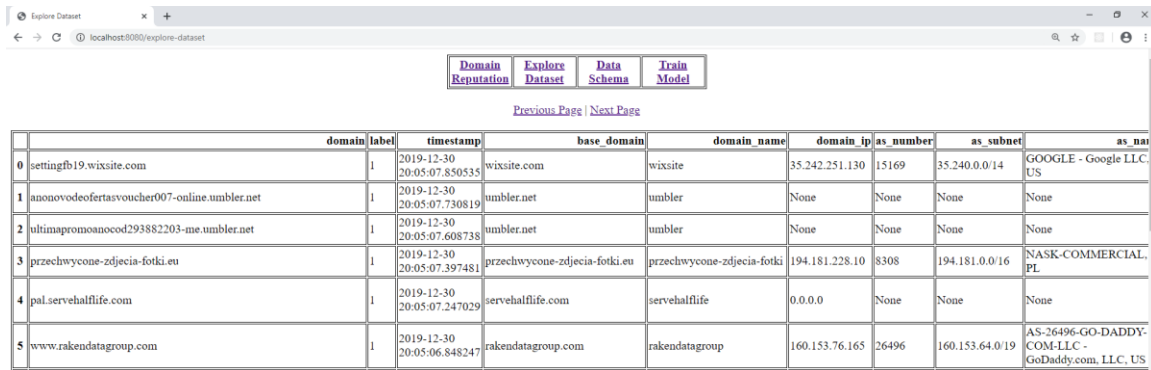


Figure 26 - SNA Ego graph with radius 2 for the domain web.whatsapp.com

5.4.2 Explore Dataset

“Explore Dataset” allows the user to browse the operational database. Each page shows 50 records, sorted by database insertion date on descending order. Figure 27 shows an output example.



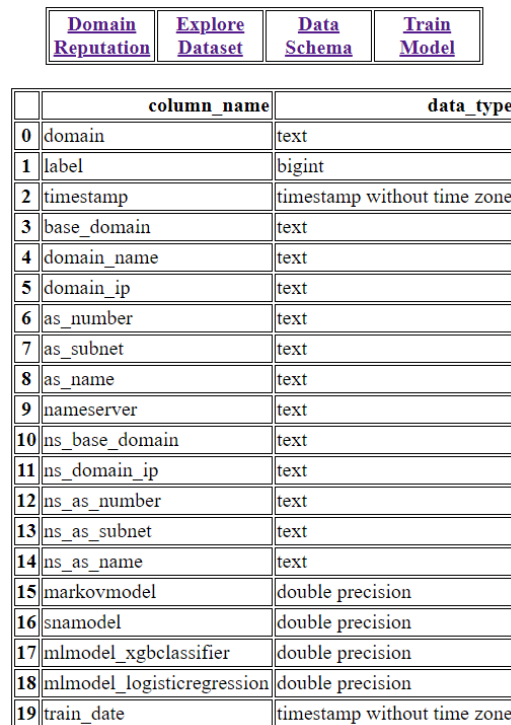
The screenshot shows a web browser window with the URL 'localhost:8080/explore-dataset'. At the top, there are four tabs: 'Domain Reputation', 'Explore Dataset' (which is active), 'Data Schema', and 'Train Model'. Below the tabs are links for 'Previous Page' and 'Next Page'. The main content is a table with the following data:

	domain	label	timestamp	base_domain	domain_name	domain_ip	as_number	as_subnet	as_name
0	settingfb19.wixsite.com	1	2019-12-30 20:05:07.850535	wixsite.com	wixsite	35.242.251.130	15169	35.240.0.0/14	GOOGLE - Google LLC, US
1	anonovodeofertasvoucher007-online.umbler.net	1	2019-12-30 20:05:07.730819	umbler.net	umbler	None	None	None	None
2	ultimapromoanocod293882203-me.umbler.net	1	2019-12-30 20:05:07.608738	umbler.net	umbler	None	None	None	None
3	przechwycone-zdjecia-fotki.eu	1	2019-12-30 20:05:07.397481	przechwycone-zdjecia-fotki.eu	przechwycone-zdjecia-fotki	194.181.228.10	8308	194.181.0.0/16	NASK-COMMERCIAL, PL
4	pal.servehalflife.com	1	2019-12-30 20:05:07.247029	servehalflife.com	servehalflife	0.0.0.0	None	None	None
5	www.rakendatagroup.com	1	2019-12-30 20:05:06.848247	rakendatagroup.com	rakendatagroup	160.153.76.165	26496	160.153.64.0/19	AS-26496-GO-DADDY-COM-LLC - GoDaddy.com, LLC, US

Figure 27 – “Explore Dataset” screenshot

5.4.3 Data Schema

“Data Schema” page allows the user to see the current database schema. Figure 28 shows a screenshot.



The screenshot shows the 'Data Schema' tab selected. At the top, there are four tabs: 'Domain Reputation', 'Explore Dataset', 'Data Schema' (which is active), and 'Train Model'. Below the tabs is a table with the following data:

	column_name	data_type
0	domain	text
1	label	bigint
2	timestamp	timestamp without time zone
3	base_domain	text
4	domain_name	text
5	domain_ip	text
6	as_number	text
7	as_subnet	text
8	as_name	text
9	nameserver	text
10	ns_base_domain	text
11	ns_domain_ip	text
12	ns_as_number	text
13	ns_as_subnet	text
14	ns_as_name	text
15	markovmodel	double precision
16	sname	double precision
17	mlmodel_xgbclassifier	double precision
18	mlmodel_logisticregression	double precision
19	train_date	timestamp without time zone

Figure 28 – “Data Schema” screenshot

5.4.4 Train Model

The “Train Model” page allows the user to train a model on the dataset with a custom amount of benign and malicious samples. Figure 29 shows an example with possible input and figure 30 shows the expected output for the same input.

Domain Reputation	Explore Dataset	Data Schema	Train Model
-----------------------------------	---------------------------------	-----------------------------	-----------------------------

File Name:	2K2K.pkl
Limit:	2000

Start Train

Figure 29 – “Train Model” screenshot

Domain Reputation	Explore Dataset	Data Schema	Train Model
-----------------------------------	---------------------------------	-----------------------------	-----------------------------

Model was trained on 2000 benign samples and 2000 malicious samples

The model is store on 2K2K.pkl

Figure 30 - example output of "Train Model"

6. Summary and Conclusions

“Average uptime of phishing attacks is around 2 days and the median uptime is only 12 hours. Due to this very short lifetime, reactive blacklisting is too slow to effectively protect users from phishing” [4]. This quote condenses the importance of this work. The phishing use-case is extraordinary from that perspective that it lives for a very short time. Therefore, a proactive approach is a clear requirement for detected threats.

The experiments described in section 4 demonstrate it's not practical to guess the domain names to be registered. A more realistic approach would be the consistently learn the internet domains' neighborhood e.g. IP, network, ASN, nameservers, etc. while doing so, constantly calculating each node's reputation. The SNA approach was proven to be very successful reaching a detection rate of 83.89% under a 1.09% false positive rate and 60.71% under a 0.35% false positive rate.

In spite of the fact, it reached a lower detection rate than PREDATOR [3] 70% detection rate given the same FPR it's a big achievement. That is because of PREDATOR leverage propriety dataset which is very expensive and takes a great deal of resources to manage. When all the classifiers developed in this project all rely only on open source data sources, and all the setup and software components described in section 5 ran on my consumer laptop. Unlike the SNA model, PREDATOR had many optimizations on the training window as shown in Table 8, where the true positive vary in the range of 58%-70%. Very close to the results of this project. Moreover, the SNA model obtains even better results than PREDATOR when considering high acceptable FPR.

a ground for future work can be to optimize the project models or create a meta-classifier that would combine the machine learning classifiers with the SNA classifier. I assume that any of the two would push the result higher than 70%.

References

- [1] Mishsky, I., Gal-Oz, N., & Gudes, E. (2015, July). A topology based flow model for computing domain reputation. In proceedings of IFIP Annual Conference on Data and Applications Security and Privacy (pp. 277-292). Springer, Cham.
- [2] Antonakakis, M., Perdisci, R., Dagon, D., Lee, W., & Feamster, N. (2010, August). Building a Dynamic Reputation System for DNS. In proceedings of USENIX security symposium (pp. 273-290).
- [3] Hao, S., Kantchelian, A., Miller, B., Paxson, V., & Feamster, N. (2016, October). PREDATOR: Proactive Recognition and Elimination of Domain Abuse at Time-Of-Registration. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (pp. 1568-1579). ACM.
- [4] Marchal, S., François, J., & Engel, T. (2012, September). Proactive discovery of phishing related domain names. In proceedings of International Workshop on Recent Advances in Intrusion Detection (pp. 190-209). Springer Berlin Heidelberg.

Appendix A – Example of Benign Domains Classification Result

domain	label	markovmodel	sname	mlmodel_xgbclassifier	mlmodel_logisticregression
datastax.com	0	0	0.226520838	0.316150874	0.152330989
omgt3.com	0	0	0.189093661	0.169931561	0.047668929
cs-gateway.cloudapp.net	0	0	0.322173269	0.563195884	0.684618425
gradientt.net	0	0	0.463472091	0.550923884	0.670320183
ipv4-c027-was001-ix.1.oca.nflxvideo.net	0	0	0.21272263	0.232552424	0.049469577
ipv4-c027-vie001-ix.1.oca.nflxvideo.net	0	0	0.21272263	0.232552424	0.049469577
api.pushe.co	0	0	0.281074716	0.2509543	0.21622384
staging.eab.com	0	0	0.197167281	0.169931561	0.064750098
vd89.mycdn.me	0	0	0.306614434	0.550923884	0.476344038
api.appmetadata.sonymobile.com	0	0	0.216388157	0.169931561	0.088441005
jtvnw.net.cdn.cloudflare.net	0	0	0.411164173	0.550923884	0.459039409
l-agent.me	0	0	0.23688687	0.169931561	0.111537037
voranda-com.videoplayerhub.com	0	0	0.404994776	0.686842501	0.728991617
fagc2-1.fna.fbcdn.net	0	0	0.415023677	0.550923884	0.634409399
cbg-app.huawei.com	0	0	0.418792725	0.550923884	0.634409399
games.geo.hosted.espn.com	0	0	0.175047154	0.169931561	0.055194319
a.smrpm.com	0	0	0.209472007	0.149449095	0.078687902
vm.mycdn.me	0	0	0.294466595	0.550923884	0.476344038
mi.walgreens.com	0	0	0.280749755	0.550923884	0.634409399
click.online.costco.com	0	0	0.27286589	0.550923884	0.415952511
nep-gw-sports.media.yahoo.com	0	0	0.32728766	0.550923884	0.343913431
s-usc1c-nss-271.firebaseio.com	0	0	0.475002479	0.771687508	0.811051469
b-cc-usea2-01-skype.cloudapp.net	0	0	0.401720014	0.563195884	0.632344189
moog-r.cyberason.net	0	0	0.18984955	0.169931561	0.051635809
keepersecurity.eu	0	0	0.173302626	0.169931561	0.057422657
link.btsvcemail.web.plus.espn.com	0	0	0.240467698	0.169931561	0.076755594
4a7b.srvng.xyz	0	0	0.314793999	0.550923884	0.354402003
s.potu.xyz	0	0	0.276672872	0.252700031	0.222190731
bs.iotleg.com	0	0	0.39703371	0.550923884	0.454590448
watsonfantasyfootball.espn.com	0	0	0.33173583	0.169931561	0.406751832
ksn.kaspersky-labs.com	0	0	0.322465091	0.550923884	0.131008276
twitter.test-app.link	0	0	0.136624822	0.169931561	0.071314612
firebat-25-aftm-80612.na.api.amazonvideo	0	0	0.163135335	0.169931561	0.054666001
instagram.fada2-1.fna.fbcdn.net	0	0	0.238939728	0.550923884	0.634409399
ios-dradis.prod.ftl.netflix.com	0	0	0.205720331	0.169931561	0.072832645
worldlifestyle.com	0	0	0.179128595	0.169931561	0.067055462
distoryrussian.info	0	0	0.360868693	0.169931561	0.467892617
failover.zingmp3.vn	0	0	0.305419921	0.550923884	0.634409399
dcs-live.apis.anvato.net	0	0	0.474100612	0.771687508	0.813777926
6xq.com	0	0	0.269454461	0.550923884	0.179828246
ssp20.pushprofit.net	0	0	0.444729318	0.550923884	0.644571404
fortnite-vod.akamaized.net	0	0	0.189559638	0.149449095	0.048604802
thepayerstribune.com	0	0	0.44605861	0.771687508	0.811051469
firebat-22-aftt-80612.na.api.amazonvideo	0	0	0.17087245	0.169931561	0.054666001
hellosubscription.com	0	0	0.227703619	0.252700031	0.222190731
tuttoabruzzo.it	0	0	0.256244965	0.252700031	0.222190731
thezeestore.com	0	0	0	0.550923884	0.634409399
rollingstone.it	0	0	0.216495485	0.169931561	0.102422666
ostetrichebrencia.it	0	0	0.300974774	0.550923884	0.603844674
mcdiscout.it	0	0	0.180040542	0.169931561	0.074450285
ixnayproductions.it	0	0	0.525489227	0.550923884	0.719478509
group-training-online.com	0	0	0.403898387	0.550923884	0.397083532
golfdom.com	0	0	0.445113739	0.686842501	0.754540184
god-games.com	0	0	0.151028405	0.550923884	0.634409399
festivalvillevesuviane.it	0	0	0.276672872	0.252700031	0.222190731
cpialegnano.edu.it	0	0	0.558343563	0.550923884	0.719478509
bartocchini.it	0	0	0.320357722	0.550923884	0.597082597
avvenire.it	0	0	0.283390057	0.550923884	0.634409399
autodirect24.com	0	0	0.151028405	0.550923884	0.634409399

Table 13 - Example of the benign domains classification result

Appendix B – Example of Malicious Domains Classification Result

domain	label	markovmodel	snamodel	mlmodel_xgbclassifier	mlmodel_logisticregression
sucursalpersonas.webcindario.com	1	0	0.796174347	0.767927349	0.839229963
inovini.com.br	1	0	0.467555451	0.231854886	0.043818797
www.safetyrd.xyz	1	0	0.77215759	0.551020205	0.808002645
bcpzonasegurabeta-viazbcp.com	1	0	0.778606534	0.551020205	0.873353206
a0375741.xsph.ru	1	0	0.852613547	0.627774358	0.969546869
ofertanatalina.store	1	0	1	0.551020205	0.634721932
salonesfloridautamu.com	1	0	0.76400821	0.659377694	0.949861811
suppottserveiteem.me	1	0	0.778583459	0.546465456	0.530266455
testsite.rebellegion.com	1	0	0.865211624	0.659377694	0.9712616
newmodelschool.org	1	0	0.823689977	0.659377694	0.797259055
treestorian.com	1	0	0.812772986	0.659377694	0.969237394
allegro.media	1	0	0.734999	0.659377694	0.730388271
www.bahianita.com	1	0	0.77275519	0.747400105	0.8972025
itokenitau.app	1	0	0.746970047	0.659377694	0.927180626
lakossagi.belepes.hu.skyorbittrading.c	1	0	0.875097843	0.835533679	0.986925721
www.cervezasorigen.com	1	0	0.819356295	0.659377694	0.943309722
zoyarentalmedan.com	1	0	0.824559586	0.659377694	0.913062758
multilinks.nuirtefrede.cf	1	0	0.727393534	0.747400105	0.938667243
com-bmnnfkppxaa.kofc3035.org	1	0	0.809575422	0.659377694	0.797259055
www.biesseacquari.com	1	0	0.734852564	0.659377694	0.973747056
khbabare2020.3dfine.com	1	0	0.693979597	0.659377694	0.934570398
instituto2005.org	1	0	0.715733205	0.659377694	0.836323606
musicaparadormir.com.br	1	0	0.835225028	0.931306899	0.992262058
golfcartbatteries.us	1	0	0.857410764	0.931306899	0.990207552
taxi-ubk.ru	1	0	0.805560159	0.659377694	0.928988732
www.ppl-vell.cf	1	0	0.843323766	0.645707488	0.921666627
updateappleidaccount.bykvijwrk.com	1	0	0.78289885	0.772657335	0.839408296
www.handrestaurant.com	1	0	0.781488571	0.931306899	0.989753181
Instagrambusinesssupport.com	1	0	1	0.659377694	0.797259055
remmancuaphuonganh.com	1	0	0.773146062	0.659377694	0.89781287
bnpparibas-mabanque.rockdelinj.com	1	0	0.784542077	0.659377694	0.963511607
mail.whistlers4hire.com	1	0	0.868753865	0.931306899	0.995028705
www.aburs.ir	1	0	0.702447716	0.659377694	0.797259055
netflipagaments.jdevcloud.com	1	0	0.880460997	0.659377694	0.972724647
fishingnewengland.com	1	0	0.756281993	0.761465013	0.884034019
emed-depot.com	1	0	0.846803284	0.659377694	0.931799398
instagram-helpconfirm.com	1	0	0.713726136	0.773697495	0.979131129
www.worldfoodinter.com	1	0	0.749305534	0.659377694	0.887738544
built4integrity.com	1	0	0.778649699	0.761465013	0.82446988
p3plvcpln1318847.prod.phx3.secureser	1	0	0.45888942	0.068464793	0.052148412
hotelcafevoud.nl	1	0	0.691529884	0.659377694	0.797259055
sherakatmarket.ir	1	0	0.721791128	0.659377694	0.80804959
vote-brexit-2020.000webhostapp.com	1	0	0.782118224	0.748818517	0.969490774
411admin.co.za	1	0	0.820172375	0.659377694	0.918700319
ebay-url.com	1	0	0.802228824	0.761465013	0.910422798
winningruby.xyz	1	0	0.78911332	0.659377694	0.961590002
kb-healthcare.com	1	0	0.885574899	0.835533679	0.986925721
castromonitoramento.com.br	1	0	0.782211813	0.931306899	0.990492108
proudcall.xyz	1	0	0.85564481	0.835533679	0.986925721
www.takilafa.com.br	1	0	0.718666955	0.659377694	0.839433614
hiersungoodresearchchemicals.com	1	0	0.795987122	0.659377694	0.934356295
unsidiomas.com.br	1	0	0.387215944	0.659377694	0.876065757
www.royalvenetian.ca	1	0	0.820633723	0.659377694	0.961765926
ecogarden.by	1	0	0.806796433	0.659377694	0.839812414
myy-proim11.com	1	0	0.85294465	0.659377694	0.960827071
mobi.facebook.com-m-ovimgntrwy.les	1	0	0.709968438	0.761465013	0.92294318
printernovin.com	1	0	0.702817494	0.659377694	0.92870676
endowmentoracle.co.kr	1	0	0.755798052	0.761465013	0.823123373
hagi-pl.com	1	0	0.784130246	0.659377694	0.898211511

Table 14 - Example of the malicious domains classification result