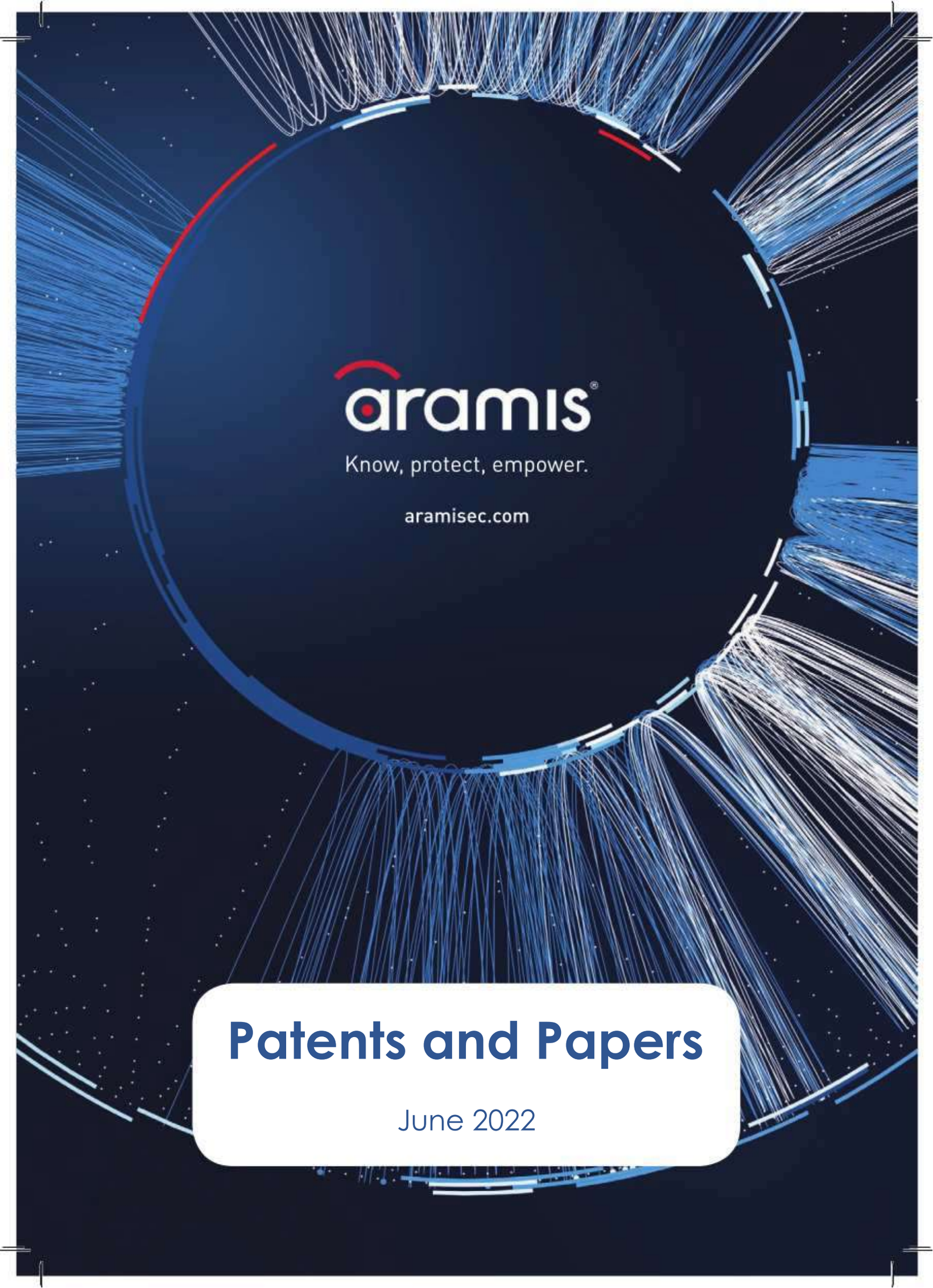


Patents and Papers

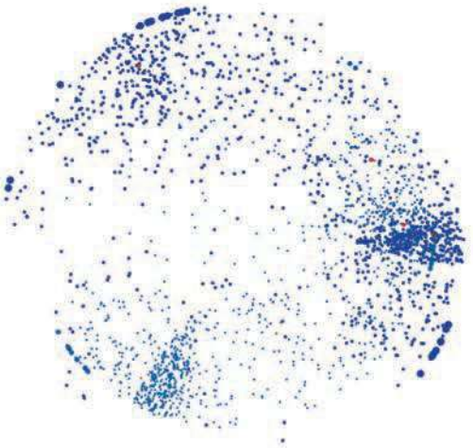
June 2022

COLLECT

Data fuels the Aramis engine.

Passive network probes are strategically positioned at various nodes within the network, depending on the throughput of the monitored flows and the amount of data transferred. Each sensor collects information from the network segment in which it is installed, analyses it in real time and sends the first results to the local server.

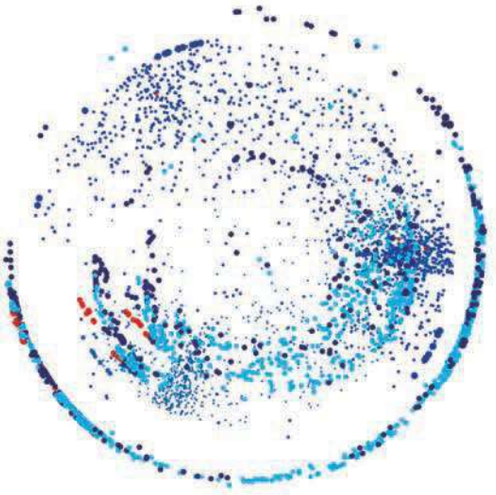
Additional: each sensor can be deployed with a "honey pot" for deception purposes.



ENRICH

Cyber Intelligence is the nitrous.

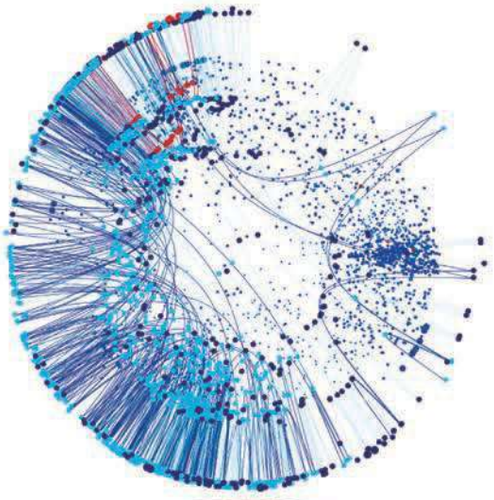
On the local server the data received from the probes are enriched with information from sources such as OSINT and Threat Intelligence as well as using information specific for the customer environment.



CORRELATE

Empower and orchestrate the mixture.

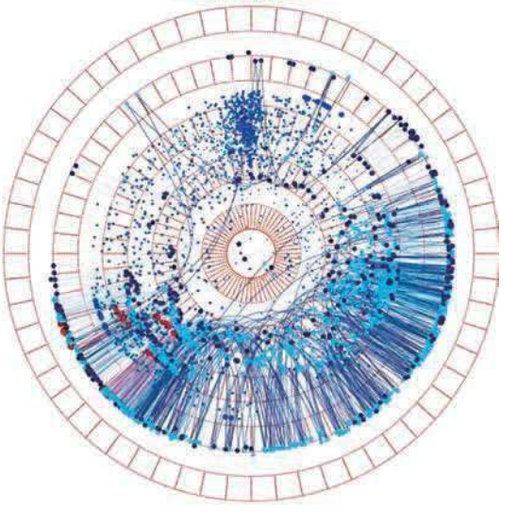
aramis constantly performs two kind of analysis on the collected data:
- Continuous Modulation of its analytics based on the dynamic variation of the measured risks.
- Analysis through the AI Engine of the behavior of each single network
node, in order to detect any possible anomaly.



VISUALIZE

Identify and act — on time.

The information is represented in the dashboards with an effective "cognitive visualization" approach allowing to promptly highlight any minimum deviations from repetitive patterns. These graphics, thanks to their zoom and drill down capabilities, afford analysts with
a powerful tool for the identification and analysis of alarms.



PATENTS

1 METHOD OF DETECTING SYSTEMATIC COMMUNICATIONS IN A COMMUNICATION NETWORK, RESPECTIVE DEVICE AND COMPUTER-PROGRAM PRODUCT

Pending Patent No.: 102021000011267
Date of Patent Application: 03/05/21

ABSTRACT

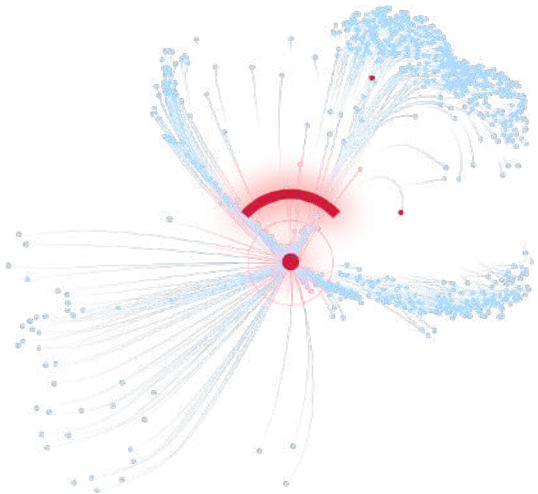
This pending patent describes a method for detecting systematic communications in a monitored network that could be indicators of a malware infection. The method analyzes specific network protocols and, for each observed packet, extracts packet metadata to measure the systematicity of transmissions between a source and a destination machine. Systematicity is computed using incremental variance to guarantee the best performance. In case a systematic communication is detected, a security operator is timely notified about the anomaly. The proposed method is implemented in an on-premise passive Security Network Monitoring Platform which collects, processes, and elaborates network flows in near-real time to detect sequences of suspicious events that probabilistically could lead back to a cyber attack.

2 METHOD OF DETECTING ANOMALIES IN SSL AND/OR TLS COMMUNICATIONS, RESPECTIVE DEVICE AND COMPUTER-PROGRAM PRODUCT

Pending Patent No.: 102021000015782
Date of Patent Application: 16/06/21

ABSTRACT

This pending patent describes a method for detecting anomalies in a monitored network regarding encrypted communications which use either SSL or TLS protocol. The proposed method analyzes SSL/TLS handshakes between network clients and external servers. For each secure connection, it extracts communication metadata regarding SSL/TLS handshakes and combines machine learning techniques with the usage of parametrized cost functions to, respectively, classify suspicious communications and attribute them an anomaly score. If a score is greater than a configurable threshold, then a security operator is notified about the occurring of a connection with a suspicious external server. The outlined method is implemented in an on-premise passive Security Network Monitoring Platform which collects, processes, and elaborates network flows in near-real time to detect sequences of suspicious events that probabilistically could lead back to a cyber attack.



3 METHOD OF DETECTING ANOMALIES IN NETWORK COMMUNICATIONS, RESPECTIVE DEVICE AND COMPUTER-PROGRAM PRODUCT

Pending Patent No.: IT102021000033203
Date of Patent Application: 31/12/21

ABSTRACT

This pending patent describes a Bayesian engine that detects anomalies in unencrypted communications of a monitored network. The proposed engine analyzes established connections between network clients and servers to detect inconsistences with respect to their usual patterns. These inconsistencies are identified through multi-stage ensemble learning techniques which allows to assign them an inconsistency score. If a score is greater than a configurable threshold, then a security operator is notified about the occurring of a suspicious communication between a client and a server. The outlined method is implemented in an on-premise passive Security Network Monitoring Platform which collects, processes, and elaborates network flows in near-real time to detect sequences of suspicious events that probabilistically could lead back to a cyber attack.

PAPERS

ITASEC

ITALIAN CONFERENCE ON CYBERSECURITY

Roma, Italy
June 20th – 23th, 2022

What is ITASEC?

The **Italian Conference on Cyber Security** is the most relevant conference dedicated to cyber security at the national level and it is organized annually by **CINI**, the **Italian National Cyber Security Lab**. The program is rich with **scientific workshops** and **tutorials**, ad hoc sessions dedicated to **academic paper** presentations, **vendor spaces** and **vision speeches** provided by sponsor companies. The main cyber security related themes include Blockchain, Cryptology, Data Security and Privacy, Security Management and Governance, Operational Incident Handling and Digital Forensics, AI and Security.

Our contribution

aizoOn presents a **scientific paper** titled *"Near-real-time Anomaly Detect-ion in Encrypted Traffic using Machine Learning Techniques"*, a joint work between the Aramis team and the aizoOn SOC.

In the last decade, the adoption of HTTPS for securing Internet communications increased by up to 90%. Threat actors adapted to this transition to HTTPS by writing more sophisticated malware that encrypt their communications with command-and-control centers. On the other hand, network security appliances are limited by the impossibility of inspecting packet payloads for deeper investigations. In this paper, we propose a cybersecurity analytics which monitors encrypted network flows and extracts features to detect possible occurring attacks and anomalies, by combining machine learning with a statistical approach. The analytics is embedded in a network security monitoring platform, named **aramis**, which provides cybersecurity analysts with a comprehensive overview of the monitored network and its traffic to support them in the identification of potentially malicious activities taking place. The detection capabilities of the proposed analytics have been tested both on a benign and a malicious dataset. This latter has been assembled by our security analysts and includes packet captures of samples and tools, respectively, developed and used by worldwide leading threat actors. Results show 96.6% accuracy on the malicious dataset, with a false positive rate approximatively equal to 0.001% when the analytics monitors legitimate encrypted network traffic.

CISDA IEEE

INTERNATIONAL IEEE SYMPOSIUM ON COMPUTATIONAL INTELLIGENCE FOR SECURITY AND DEFENSE APPLICATIONS (IEEE CISDA)

Virtual Event

December 4th-7th, 2021

IEEE SSCI 2021

What is CISDA?

CISDA is a symposium that brings together industry practitioners and researchers, together with academicians, with the aim of **presenting current and ongoing efforts in computational intelligence to detect and adapt to emerging threats**. This year, CISDA presentations will tackle the analysis of the main **challenges to solve problems in the fields of security and defense** with the application of novel techniques, ranging from **neural networks** to **evolutionary computation** and **swarm intelligence**. The symposium is held in conjunction with other symposia, annually organized during the IEEE Symposium Series on Computational Intelligence (IEEE SSCI), in the effort to **promote and stimulate discussion on the latest theory, algorithms, applications and emerging topics on computational intelligence**.

Our contribution

aizoOn presented a **scientific paper** titled *"Near-real-time Anomaly Detect-ion in Encrypted Traffic using Machine Learning Techniques"*, a joint work between the Aramis team and the aizoOn SOC.

In the last decade, the adoption of HTTPS for securing Internet communications increased by up to 90%. Threat actors adapted to this transition to HTTPS by writing more sophisticated malware that encrypt their communications with command-and-control centers. On the other hand, network security appliances are limited by the impossibility of inspecting packet payloads for deeper investigations. In this paper, we propose a cybersecurity analytics which monitors encrypted network flows and extracts features to detect possible occurring attacks and anomalies, by combining machine learning with a statistical approach. The analytics is embedded in a network security monitoring platform, named **aramis**, which provides cybersecurity analysts with a comprehensive overview of the monitored network and its traffic to support them in the identification of potentially malicious activities taking place. The detection capabilities of the proposed analytics have been tested both on a benign and a malicious dataset. This latter has been assembled by our security analysts and includes packet captures of samples and tools, respectively, developed and used by worldwide leading threat actors. Results show 96.6% accuracy on the malicious dataset, with a false positive rate approximatively equal to 0.001% when the analytics monitors legitimate encrypted network traffic.

Near-real-time Anomaly Detection in Encrypted Traffic using Machine Learning Techniques

Daniele Ucci, Filippo Sobrero, Federica Bisio
Data Analytics Team
Cyber Security Division
aizoOn Technology Consulting
Turin, Italy
{name}.{surname}@aizoongroup.com

Matteo Zorzino
Intelligent Security Operation Center
Cyber Security Division
aizoOn Technology Consulting
Turin, Italy
{name}.{surname}@aizoongroup.com

***Abstract*—In the last decade, the adoption of HTTPS for securing Internet communications increased by up to 90%. Threat actors adapted to this transition to HTTPS by writing more sophisticated malware that encrypt their communications with command-and-control centers. On the other hand, network security appliances are limited by the impossibility of inspecting packet payloads for deeper investigations. In this paper, we propose a cybersecurity analytics which monitors encrypted network flows and extracts features to detect possible occurring attacks and anomalies, by combining machine learning with a statistical approach. The analytics is embedded in a network security monitoring platform, named aramis®, which provides cybersecurity analysts with a comprehensive overview of the monitored network and its traffic to support them in the identification of potentially malicious activities taking place. The detection capabilities of the proposed analytics have been tested both on a benign and a malicious dataset. This latter has been assembled by our security analysts and includes packet captures of samples and tools, respectively, developed and used by worldwide leading threat actors. Results show 96.6% accuracy on the malicious dataset, with a false positive rate approximatively equal to 0.001% when the analytics monitors legitimate encrypted network traffic.**

***Index Terms*—encrypted malware communications, passive network analysis, anomaly detection, machine learning, SSL, JA3**

I. Introduction

Nowadays the vast majority of Internet traffic is encrypted thanks to a cross-industry effort involving companies both from private and public sector. This effort started in the '90s but, only in recent years, the percentage of HTTPS encrypted network traffic has experienced a significant increase [1], up to achieving a percentage ranging between 80% and 90% [1]–[5]. Clearly, encrypted communication adoption varies from country to country and may increase quickly in some regions with respect to others [5].

The implications are twofold: on the one hand, threat actors adapted to the transition from HTTP to HTTPS, at a higher economic cost, performing more sophisticated and concealed attacks; on the other hand, network security appliances are limited by the impossibility of inspecting packet payloads for deeper investigations. The combination of these two factors, in 2020, enabled threat actors to perform malware campaigns relying on HTTPS for delivering malware, contacting command-and-control activity, and exfiltrating data [6]. In particular, just in 2020, 67% of malware has been delivered via encrypted HTTPS connections [7]. In addition, data exfiltration and sensitive information stealing have always represented a challenging threat for companies [8]–[10], primarily from a financial point of view [6]. With the mainstream adoption of secure communications (also used by attackers), specific countermeasures need to be taken into account.

Both academia and industry have proposed different solutions to cope with encrypted traffic, as discussed more in detail in Section II of this paper. However, a key point that differentiates the various approaches is their level of intrusiveness: some approaches work directly with encrypted traffic, while others decrypt and re-encrypt data to be inspected. The first ones do not decipher encrypted communications, but consider exchanged data and metadata. For this reason, these approaches are not able to detect compliance and policy violations or possible security breaches by examining traffic payloads. Conversely, there exist approaches implemented in security products, like [11], which decrypt secure communications and allow to analyze payloads. However, decryption and encryption processes insert a significant computational overhead that negatively impacts the performance of these security products [4]. As discussed later in the paper, the protocols on which HTTPS relies on provide many negotiable cipher suites that are not necessarily supported by a specific security product [4]: according to [6], 60% of organizations is not prepared to decrypt HTTPS traffic efficiently.

In this context, we propose an advanced cybersecurity analytics (ACA) which analyzes HTTPS exchanged protocol messages and extract data and metadata to detect possible occurring attacks and anomalies. More in detail, the ACA extracts metadata contained in the fields of X.509 certificates and SSL/TLS metadata and has been designed to detect anomalies taking place during a SSL/TLS handshake between a client and an external server. The analytics combines an unsupervised machine learning technique with a statistical approach: after characterizing the SSL/TLS flow with selected features, a machine learning module isolates anomalous connections and an anomaly score is calculated in order to alert security analysts about potential malicious communications.

The proposed algorithm is embedded in aramis® (Aizoon Research for Advanced Malware Identification System), a commercial network security monitoring platform able to collect, process, and elaborate network flows in near-real time in order to detect and investigate potential malicious or anomalous activities. Network data are processed to detect potentially malicious activities and, in case of successful detection, two different kinds of notifications can be issued to SOC analysts: the first one involves the observation in the network traffic of one, or more, indicators of compromise, while the second type of alerts comes from aramis®' ACAs. Each ACA is a combination of different statistical approaches and unsupervised machine learning algorithms. Starting from these alerts, analysts can rely on the platform's dashboards, that offer drill-down capabilities, to further investigate alert notifications by: correlating alerts produced by other analytics (e.g., detection of malicious payload downloads from compromised sites), or analyzing similar behaviors throughout the monitored network (e.g., machines sharing the same user agent or contacting the same command-and-control center).

The rest of the paper is organized as follows: Section II discusses related work, while Section III introduces basic notions that will be later used to detail the proposed approach (Section IV). The experimental evaluation is reported in Section V and Section VI presents a real-world case study. Finally, Section VII concludes the paper.

II. Related Work

The use of encryption poses significant challenges to network threat detection due to the inapplicability of traditional signature-matching techniques and the increasing number of malware authors taking advantage of it, as outlined in Section I.

The security community has therefore researched in two main directions: decryption of traffic flows [11], [12] and use of network-flow-based metadata [13]. Since decrypting network traffic and applying traditional signature based approaches to detect cyber threats is not always possible, not only due to privacy and legal concerns, but also for the introduced considerable overhead (as discussed in Section I), the combination of passive data extraction from a monitored network and subsequent application of machine learning techniques on SSL/TLS metadata has more and more become an appealing solution [6], [14]–[16]. As an example, [17] performed an analysis over millions of SSL/TLS encrypted flows and a study on 18 malware families by extracting meaningful features from data. With the widespread use of machine learning techniques, research focus has hence moved on the feature engineering tasks [18], [19]. Two different groups of features may be currently found in literature: statistical and sequential features. Statistical features contain but are not limited to flow-level metadata, packet length distributions, time distributions, byte distributions and SSL/TLS header information [17]. An example of deep learning framework combining statistical features can be found in [19]. Sequential features are obtained from the raw flow sequences by learning the generation probabilities of flows. By representing the traffic flow sequence via Markov transformation matrix, [20] clustered certificate lengths and first packet lengths to improve the classification performance under a second-order Markov model.

The approach we present and evaluate in the next sections passively extracts both statistical and sequential features from network flows to detect anomalies in a monitored network. Differently from [20], we leverage machine learning to recognize SSL/TLS handshakes deviating from the ones usually established in the network. Similar to [6] the proposed analytics establishes a baseline of usually secure connections, by using a different set of features.

III. Background

A. SSL and TLS protocols

SSL and TLS protocols allow two machines to authenticate and establish a session key, created to cryptographically protect the remainder of the session [21]. Authentication is performed by means of certificates, which are signed messages reporting the identity of either an individual, a host, or an organization. In the World Wide Web, certificates are typically signed by trusted nodes, called Certification Authorities (CA). The standard used for SSL/TLS protocols to define public key certificates' format is X.509, version 3 [22]. The advanced cybersecurity analytics we propose in this paper analyzes a subset of all the available fields in X.509 certificates, that are briefly described in the following. The signature field contains both the algorithm identifier and hash function used by the CA for signing the certificate (e.g., sha-1WithRSAEncryption). On the other hand, the validity field stores the time interval during which the CA ensures that it will keep information about the certified entity, specified in the subject name field. Each X.509 certificate contains, respectively, information about the subject public key and the issuer: the first specifies the public key itself and the algorithm applied for generating it (e.g., rsaEncryption), while the second reports the name of the CA that issued the public-key certificate.

B. One-class SVM

The original formulation of Support Vector Machines (SVMs) is related to the resolution of supervised tasks, but the one-class SVM has been shown to represent a suitable choice in the context of anomaly detection [23]. It is defined as a boundary-based anomaly detection method, which modifies the original SVM approach by extending it in order to deal with unlabeled data. Like traditional SVMs, one-class SVMs can also benefit of the so called kernel trick when extended to non-linearly transformed spaces, by defining an appropriate scalar product in the feature space.

C. Jenks' natural breaks optimization

This optimization method, applied to power-law distributions, divides input instances in classes by minimizing within-class variance, while maximizing between-class variance [24]. The goodness-of-variance-fit (gvf) value expresses the divergence between predicted classes and observed values. Jenks'

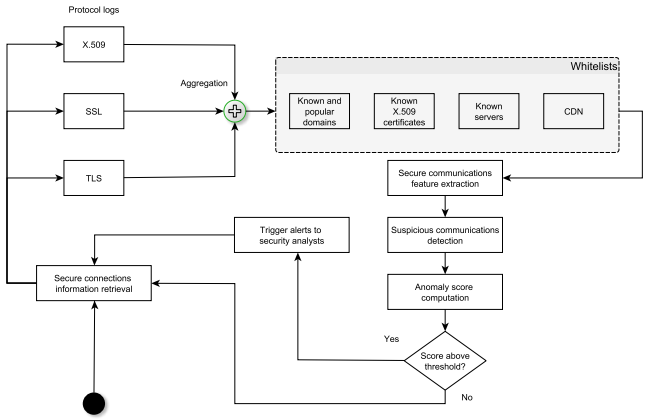


Fig. 1. SSL/TLS analytics overview.

natural breaks optimization consists in iteratively computing the gvf by moving one data value from the class with the largest deviations from the mean to the class with the lowest ones, until the sum of the within-class deviations reaches a minimum [25].

IV. SSL/TLS Analytics

The proposed approach aims at detecting possible anomalies occurring during a SSL/TLS handshake between a client, located inside the network monitored by the software platform outlined in Section I, and an external server. We recall that SSL/TLS protocols enable two machines to securely communicate over an unprotected network (e.g, Internet), as mentioned in Section III-A.

Detection of possible anomalies may be performed by simply analyzing the information exchanged during SSL/TLS handshakes, e.g., by examining the issuer and subject fields of a certificate. Another element that requires particular attention is represented by self-signed X.509 certificates: in this case, the issuer and the subject fields share the same CA value, and the private key employed by the CA to sign the certificate corresponds to the public key certified within the certificate itself [22]. The challenge is here represented by the fact that self-signed certificates can be included in certification paths and can be legitimately used by CAs to advertise information about their operations. However, it is an ever-growing common practice for malware to communicate with their command-and-control servers using a self-signed certificate.

Therefore, the SSL/TLS detection analytics examines information contained in X.509, SSL, and TLS exchanged protocol messages. As mentioned in Section I, aramis® is designed to collect data and metadata related to all the packets transmitted in the monitored network. After data collection, aggregation, and filtering, the SSL/TLS analytics extracts, for each SSL/TLS flow, features able to capture possible anomalies in the communication. Selected features are fed to a machine learning module, which detects suspicious connections, whose anomaly score is eventually computed and possible alerts are signaled to security analysts.


```
{
  "version" : "TLSv12",
  "server_name" : "teams.microsoft.com",
  "curve" : "secp384r1",
  "subject" : "CN=teams.microsoft.com",
  "issuer" : "CN=Microsoft RSA TLS CA 01,
              O=Microsoft Corporation,C=US",
  "server_cert_chain" : [
   {
    "md5" : "28211f1f8a50966b518ec39d3546d57d",
    "sha1" : "4a263f1f39dd526901987ecdb09e2d1297e2bc51",
    "x509" : {
        "version" : 3,
        "key_type" : "rsa",
        "key_alg" : "rsaEncryption",
        "key_length" : 2048,
        "sig_alg" : "sha256WithRSAEncryption",
        "not_valid_before" : 1606847889.0,
        "not_valid_after" : 1638383889.0,
        "subject" : "CN=teams.microsoft.com",
        "issuer" : "CN=Microsoft RSA TLS CA 01,
                    O=Microsoft Corporation,C=US",
      }
    }
  ],
  "ja3" : "7f805430de1e7d98b1de033adb58cf46",
  "ja3s" : "0f14538e1c9070becdad7739c67d6363",
  "cipher" : "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384",
  "machineDest" : "TEAMS.MICROSOFT.COM"
}
```


Fig. 2. Sample of a communication log including both TLS and X.509 information. For space constraints, in the server certification chain we kept only the end-user certificate.

A. General approach and feature extraction

Figure 1 reports the general structure of the proposed detection method: network traffic involving secure connections is monitored, collected and stored in a database. This knowledge base is periodically accessed in order to retrieve updated information about established encrypted outbound connections, i.e., data and metadata about X.509, SSL, and TLS exchanged protocol messages.

Data and metadata related to the same communication between a client and a server are then aggregated: indeed, a communication is eventually described by complementary information given by both SSL/TLS and X.509. Figure 2 shows an example of an aggregated log comprising both TLS and X.509 information, related to a secure connection established with the business communication platform Microsoft Teams[1]. For privacy reasons, we omitted from the log all the sensitive information as, for example, the IP addresses involved in the communication. It is important to note that each log contains a subset of the information briefly discussed in Section III-A.

The filtering phase allows to remove from input data information about connections to known and popular domains, servers, CDNs, and X.509 certificates trusted by the company where aramis® is deployed. It is worth noting that the platform itself enriches the whitelists database by taking trace of popular secure connections and highly visited servers,

[1]Microsoft Teams:https://teams.microsoft.com/

TABLE I
LIST OF NUMERIC FEATURES EXTRACTED FROM SSL/TLS FLOWS

Feature ID	*Numeric features F_n*
n_0	JA3 popularity (see IV-B for further details)
n_1	Server certificate chain popularity (see IV-B for further details)
n_2	Number of self-signed certificates normalized over a value indicating the maximum length of a certificate chain (e.g., 100)
n_3	Number of expired certificates normalized over a value indicating the maximum length of a certificate chain (e.g., 100)
n_4	Number of certificates reporting an anomalous validity (e.g., a validity less than 3 days) normalized over a value indicating the maximum length of a certificate chain (e.g., 100)
n_5	Number of certificates signed with a weak signing algorithms normalized over a value indicating the maximum length of a certificate chain (e.g., 100)

TABLE II
LIST OF BOOLEAN FEATURES EXTRACTED FROM SSL/TLS FLOWS

Feature ID	*Boolean features F_b*
b_0	The server certificate (or a certificate stored in the server certificate chain) is self-signed
b_1	The certificate signed by the server (or a certificate stored in the server certificate chain) is expired
b_2	The subject contained in the end-user certificate has an invalid top-level domain
b_3	The country listed in the end-user certificate is not valid
b_4	One of the certificates in the server certificate chain has an anomalous validity (e.g., a validity less than 3 days)
b_5	One of the certificates in the server certificate chain relies on a weak signing algorithm
b_6	The server name is not a sub-domain of the end-user subject's certificate
b_7, b_8, b_9	The server name, the subject, and the issuer of the end-user certificate might be randomly generated (see IV-C for further details)

through two different signatures: 'JA3' hashes and server certificate chains. A JA3 hash is defined as a fingerprint of a SSL/TLS flow generated by a client, built from the following handshake information: SSL/TLS protocol version, type of employed cypher, possible extension values [26], enumeration of the supported elliptic curves, and the point formats such curves can parse [27]; hexadecimal values representing this information are concatenated and then hashed trough an MD5 function. On the other hand, server certificate chains allow to identify communications with specific servers, encoded using MD5 hashes: each SHA1 hash identifying a specific certificate in the chain is concatenated with the other chain's SHA1s, which identify the other certificates in the chain. Concatenated SHA1s are then hashed using an MD5 function.

In order to create the feature space to be used by the machine learning algorithm and analytics' modules, the SSL/TLS analytics extracts, for each SSL/TLS flow, both numeric and boolean features which are listed in Table I and II. These chosen features are able to capture signals indicating possible anomalies in the certification chain sent by server to the client.

B. Popularity calculation

Regarding the JA3 and server certificate chain popularity (features n_0 and n_1 in Table I), term frequency metrics calculation is applied [28]. In particular, both the JA3 hashes and the received server certificate chains are generalized as terms $t_0, \ldots, t_i$, respectively generated and exchanged during SSL/TLS handshakes in a predefined time interval Δt. As an example, one can compute the popularity of a specific JA3 hash h, using one of the standard term-frequency tf definitions, commonly applied in information retrieval:

$$tf(h) = \frac{c_h}{\sum_{h' \in H} c_{h'}} \quad (1)$$

where c_h is the number of occurrences of h divided by the total number of hashes $|H|$ observed in Δt. Analogously, it is possible to compute the popularity of a server certification chain scc through $tf(scc)$. Popularity values are normalized and used to filter out either popular secure connections or highly visited servers.

C. Randomness calculation

Randomness of the server name, the subject, and the issuer of the end-user certificate (used to extract features b_7,b_8 and b_9 in Table II) is intended to measure how much their monograms and bigrams characters distributions are "close" to the ones associated to randomly generated strings, according, for example, to the English language characters distribution (note that also other languages can be configured). Such measure is obtained by employing the same approach of an open-source random string detector based on a 2-character Markov chain [29]: this system is trained on pairs of subsequent characters extracted from few megabytes of English text to let the Markov chain learn which is the probability distribution of the appearance of a character following a given one. Thus, the trained Markov chain fed with an input string produces an output probability p indicating how much the string follows the language distribution: the resulting probability grows as the similarity to English words gets higher.

D. Anomaly detection

Numeric features and a subset of boolean features – namely $F_{\text{SVM}} := \{F_n \cup b_7, b_8, b_9\}$ – are given in input to the one-class classifier described in Section III-B. In particular, we rely on a R library implementation of the one-class SVM that uses a radial basis function kernel. SVM hyperparameters ν and γ have been tuned to minimize generalization error, as discussed in [30], and respectively set to 0.5 and 0.1. Moreover, we remove from F_{SVM} all those features whose variance is equal to 0, because they do not add any information to the built model. The model creation phase [31] is performed on the collected historical data, which compose the *training set* used

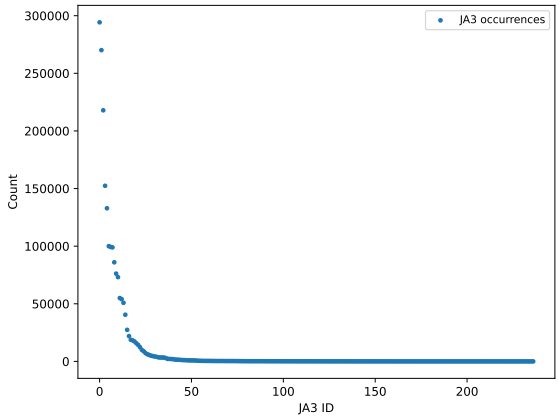


Fig. 3. Distribution of JA3 hash occurrences in the monotored network, observed during a period of 20 days. For privacy reasons, on the x axis, identifiers associated to unique JA3 hashes are reported instead of actual hashes.

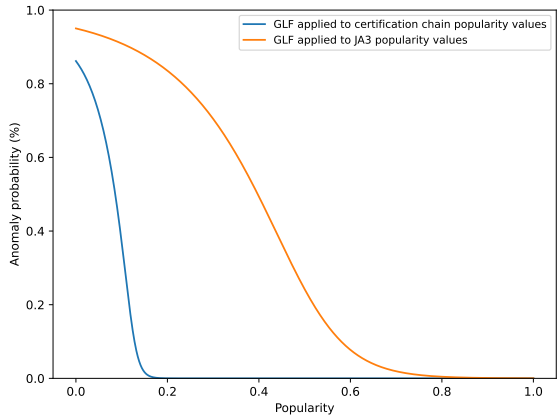


Fig. 4. Example of two generalized logistic functions that convert popularity values into anomaly probabilities. The two functions have been respectively obtained using the following parameters: $K = 0.95$, $S = 0$, $C = Q = 1.0$, $x_0 = 0.125$, $B = 95$, $\nu = 5$ and $K = 1.0$, $S = 0$, $C = Q = 1.0$, $x_0 = 0.5$, $B = 15$, $\nu = 2.5$.

to train the model. Models are periodically updated using 6 hours of secure traffic data. Outputs therefore represent secure connections which deviate – in one or more features – from the ones normally observed in the monitored network. Built machine learning models can be also helpful in detecting zero-day attacks, because malicious traffic samples are not required for the learning phase.

It is worth noting that connections identified as anomalous by the classifier are not necessarily malicious: for example, legit self-signed company certificates are correctly detected by the SVM module as anomalous, but they do not represent a cyber threat. To reduce false positives and be sure of analyzing only suspicious communications that, hence, are not widespread across the monitored network, the analytics filters out popular JA3 hashes. Since JA3 frequencies follow the power law distribution, as shown in Figure 3, we apply the Jenks' natural breaks optimization (described in Section III-C) to extract the least popular JA3s and take into account only rare JA3 hashes. By employing a variant of an R open-source algorithm for computing Jenks' natural breaks [32], the analytics first calculates which is the optimum number of breaks that allows to achieve the maximum goodness-of-variance-fit. Then, by running again the optimization method and providing the optimum number of breaks to divide input hashes in classes, the analytics selects all the JA3 hashes included in the last class (i.e., the least popular). At this point, all the hashes having a term frequency (see Equation 1) higher than the one of the least popular JA3 hashes are filtered out. After filtering, the remaining term frequencies are normalized by dividing them by the maximum observed frequency.

As shown in Figure 1, flows detected as suspicious by the SVM and whose JA3 hashes are not popular are fed to a second analysis module that computes an anomaly probability A_F. The module combines configurable weights and two different parametric generalized logistic functions, which both allow to tune A_F according to the needs of SOC analysts. In other words, a weight w_i is associated to each one of the boolean features F_b listed in Table II and an anomaly probability A_F is computed using a weighted arithmetic mean: $A_F = \sum_{i=1}^{|F_b|} w_i \cdot b'_i$, where b'_i represents the result of the conversion of a boolean value into either a 0 or 1, depending on its truth value (i.e., 0 if false, 1 otherwise). Each weight w_i is adjusted according to the nature of the traffic analyzed by aramis®[2]. As an example, a SOC analyst may set a higher weight to the feature representing whether the end-user certificate is self-signed: if the monitored network does not employ self-signed certificates, then the security analyst may assign a higher weight to this feature in order to spot possible connections relying on a self-signed certificate.

On the other hand, JA3 and server certificate chain popularities are converted to anomaly probabilities using two generalized logistic functions of the form

$$K - \frac{K - S}{(C + Qe^{-B(x-x_0)})^{1/\nu}}$$

This type of functions was initially proposed by Richards in 1959 to model plants growth rate [33], but it turned out to be able to generate more flexible S-shaped curves. In the above equation x are rank values, K is the upper asymptote, S is the lower asymptote value, C, Q, and x_0 are proper constant values, B is the growth rate, and ν affects the direction along which maximum growth occurs. Figure 4 shows an example of two logistic functions, deployed to production, in which the first one has been specifically designed to filter out known and/or highly visited external servers (i.e, whose certification chain popularity is greater or equal to 15%) by assigning them an anomaly probability equal to 0.

Anomaly probabilities resulting from the application of logistic functions are first averaged between them, and then with A_F to obtain the final anomaly probability A. Finally,

[2]Weights w_i reflect the importance given by SOC analysts to the corresponding features b'_i; for the experimental evaluation, assigned weight values are $w_0 = 0.2$, $w_1 = w_4 = w_5 = w_6 = 0.13$, $w_2 = w_3 = 0.005$, $w_7 = w_8 = w_9 = 0.09$.

TABLE III
MALWARE PACKET CAPTURES SUMMARY

Type	*Family*	*Threat actor*	*Domain*	*Year*	*Stage*	A_m
Banking trojan, info stealer	Emotet	Mummy Spider, Mealy Bug	womenempowermentpakistan.com, fynart.com, www.laminatedtube.com, ygpryd.com, thammynhp.com, shop.homenhealthy.com, rocketviral.com, www.campuscamarafp.com, snjwellers.com, pesquisacred.com, theaffiliateincome.com, stars-castle.ir, travianbot.net, lamajesteindustries.com, nanettecook.org	2020-2021	Dropper loading	0.48
Info stealer, banking malware	Ursnif	Golden Cabin	-	2021	Callback	0.63
Info stealer, banking malware	IcedID	Gold Cabin	marslayot.top, garrozalibbo.click	2021	Callback	0.70
Info stealer, banking malware	Bazarloader	UNC1878	-	2021	Callback	0.60
Banking trojan	Trickbot	Graceful Spider, UNC1878, Wizard Spider	api.ip.sb, barionexis.top, ident.me, api.ipify.org, liverpooldabestteamoftheworld.com	2021	Information gathering, dropper loading, and data exfiltration	0.62
Info stealer	Lokibot	Sweed, The Gorgon Group	makiyazhdoma.ru, itsssl.com, hiokurl .com, hyp.ae, pxlme.me bakercost.gq	2021	Dropper Loading	0.49
Info stealer	AgentTesla	Sweed	api.ip.sb, iplogger.[ru|org], ipinfo.io, 2no.co, connectini.net, [alb].xyzgame.cc, fb.xiaomishop.me, spark.lightburst.xyz, ezps.co.uk, shadow-vpn.net, iplis.ru, p6701.softemstore.xyz, 2no.co, www.profitabletrustednet9work.com, bucket.swiftlaunchx.com	2021	Information gathering and dropper loading	0.70
Info stealer	AZORult	The Gorgon Group	tradecontract.es, telete.in, iplogger.org, music-s.xyz	2021	Dropper loading	0.41
Remote Access Trojan	Jssloader	FIN7	injuryless.com	2021	Dropper loading	0.41
Remote Access Trojan	AsyncRAT	-	-	2021	Dropper loading	0.63
Dropper	Chopstick	APT28	cdnverify.net, mvband.net	2020	Dropper loading	0.81
Dropper	Gamaredon Downloader	Gamaredon	hastebin.com religonclothes.com	2020	Dropper loading and callback	0.50
Coin miner	Coinminer	-	iplogger.org	2020	Information gathering	0.51
Ransomware	GandCrab	-	www.billerimpex.com	2018	Callback	0.60

TABLE IV
TOOL PACKET CAPTURES SUMMARY

Tool	*Threat actor*	*License*	*Capabilities*	A_m
Empire	CopyKittens, FIN10, APT19, APT33, Turla, Wirte, Silence, Frankenstein, Wizard Spider, Muddy Water, APT41, Indrik Spider	Open-source	Remote administration and post-exploitation framework	0.69
Cobalt Strike	APT 29, APT32, APT41, Anunak, Cobalt, Codoso, CopyKittens, DarkHydrus, FIN6, Leviathan, Mustang Panda, Shell Crew, Stone Panda, UNC1878, UNC2452, Winnti Umbrella	Commercial	Penetration testing product that allows an attacker to deploy an agent on the victim machine	0.51
Meterpreter[3]	-	Open-source	Attack payload that provides an interactive shell from which an attacker can explore the target machine and execute code	0.60

A is further increased by using a configurable weight w_N ($w_N = 0.25$ in the experimental evaluation reported in Section V) if any of the features observed in the test set has not been seen before: as mentioned in Section IV-D, we filter out features in F_{SVM} whose variance is 0 and, for this reason, it may happen that some features observed in the test do not appear in the trained model. As an example, aramis® may not have seen during the SVM training time an expired certificate; thus, if the analytics observes a secure communication involving an expired certificate, then such anomaly is addressed by increasing the overall anomaly probability A by using w_N. If the anomaly probability A is above a configurable threshold A_{th}, then the security analysts are notified about suspicious secure connections established with external servers located outside the monitored network. A_{th} can be dynamically configured to provide in output only anomaly probabilities approximately greater than 16%, 30%, 44%, and 62%, according to SOC's analysis requirements.

V. EXPERIMENTAL EVALUATION

For the experimental evaluation of the proposed approach, we collected both a malicious and a benign dataset. The first one is constituted by packet captures generated from samples and tools, respectively, developed and used by worldwide leading threat actors [7], [34]–[36], which have been injected in aramis® in order to be processed and analyzed as ordinary traffic. Packet captures of samples and tools have been selected by our security analysts from three online malware analysis services, namely ANY.RUN[4], Hybrid Analysis[5] and

[3]Even though Meterpreter is not a tool used by threat actors, it allows to craft fake self-signed certificates

[4]ANY.RUN: https://app.any.run/

[5]Hybrid Analysis: https://www.hybrid-analysis.com/

TABLE V
LEGITIMATE NETWORK TRAFFIC DATASET SUMMARY

Statistics	*Count*	*One-hour mean*
Total number of SSL/TLS logs	2M	3,478
Total number of machines	578	58
Total number of unique server names	19,421	344
Total number of unique JA3	233	39
Total number of unique JA3s	487	80
Total number of unique cert. chain SHA1s	7,561	257

Malware Traffic Analysis[6]. Chosen samples contain, among legitimate network traffic, SSL/TLS malicious communications. Tables III and IV report a summary of the malicious assembled dataset: for malware, we indicate type and family while, for tools, we report their licenses and corresponding tool capabilities. In both cases, we detail threat actors according to the definitions given by MITRE[7] and Malpedia[8]. Analogously to FireEye [37], we listed also malware stages to better characterize and describe collected samples: dropper loading, callback and data exfiltration. In the majority of analyzed samples, SSL/TLS connections have been established for downloading droppers and contacting command-and-control centers. In addition to the stages described by FireEye, we introduce the information gathering stage, that represents a phase in which malware leverage external services to investigate and reveal external IPs of compromised machines. On the other hand, the benign communication dataset gathers legitimate SSL/TLS communications observed in a real corporate network during a period of 24 days. Table V summarizes general statistics about the network traffic taken into account. Both datasets have been injected in a controlled network environment provided with aramis®, the network security monitoring platform briefly discussed in Section I, to respectively measure the accuracy of the proposed approach and to evaluate the false positive rate on legitimate secure connections. Performed evaluations have been split because the malicious dataset is divided in samples, while the benign one in connections. Hence, depending on the input dataset, quantities of true and false positives and true and false negatives are, respectively, referred to either malware samples or benign connections. As an example, for the malicious dataset, true positives are defined as the number of correctly detected malware/tool samples, while false negatives as the number of malware/tool samples incorrectly labeled as benign. In this case, false positives and true negatives are equal to 0 because no benign sample is included in the malicious dataset. Analogous considerations apply to the benign dataset.

The accuracy measured on the malicious dataset, which includes 59 samples collected by our security analysts, is equal to 96.6%. On the other hand, the false positive rate calculated on the benign network traffic is approximately equal to 0.001%. Tables III and IV report in column A_m the mean of maximum anomaly probabilities computed when analyzing malicious samples. It is worth noting that, even if their values are not high, anomaly probabilities of legitimate communications are usually below these percentages, setting at an average of 40% with a standard deviation of 0.13, which results in $\approx 70\%$ of the false positives having an anomaly probability lower than malicious samples.

Regarding accuracy, among all samples, only two packet captures of IcedID family have not been successfully detected by our analytics. After an investigation, we found out that the JA3 generated by malware samples (i.e., a0e9f5d64349fb13191bc781f81f42e1) collides with several hashes produced by legitimate and common services, like Google APIs and Microsoft Office. This is a known downside of JA3 hashes [38] which may collide with hashes belonging to legitimate applications and prevents the detection of malicious samples. Nevertheless, we are still able to identify malicious encrypted communications performed by malware developed by some of the most famous threat actors. Concerning the false positive rate, it is important to note that the obtained value allows SOC analysts to proceed with deeper investigations avoiding unprocessable amounts of alert notifications.

VI. DISCUSSION

As mentioned in Section I, the proposed analytics is embedded in a network security monitoring platform able to support SOC analysts in investigating cyber threats, as presented further in this section. As an example, we analyze packet captures generated by a malware sample of the family Astaroth, a trojan and an information stealer widely known to attack companies in Europe and Latin America since late 2017 [39]. In July 2021, samples – like the one considered in this section – have been distributed in a Malspam campaign through mails containing an infected link. By accessing this link, the user downloads a compressed Powershell script that, if executed, performs its malicious activity in different infection stages: first, it creates a support file in the public "Videos" folder in which specifies a command-and-control center (C&C) domain; then, the script establishes an HTTP connection with this latter to download an XML file containing Javascript source code, which is later run to download 3 malicious DLLs and a compiler for guaranteeing persistance. Once the malicious DLLs are loaded, they extract sensitive information (e.g., mail, e-commerce and banking accounts) from the victim machine that, afterwards, are exfiltrated to either predetermined or algorithmically generated malicious domains.

aramis® has been able to successfully detect as malicious the downloads of the 3 DLLs and the compiler through an ACA that analyzes HTTP traffic. In parallel, DNS algorithmically generated requests have been detected by two other analytics, responsible for monitoring DNS traffic [40], [41]. Finally, the encrypted communications with a C&C have been correctly identified by the analytics presented in this paper. Interestingly, aramis® did not detect exfiltrations through HTTP with any ACA but, thanks to the platform monitoring capabilies, SOC analysts have been able to identify HTTP requests exfiltrating data, as reported in Figure 5. Our SOC analysts speculate that third level domains, depicted in

[6]Malware Traffic Analysis: https://www.malware-traffic-analysis.net/
[7]MITRE: https://attack.mitre.org/
[8]Malpedia: https://malpedia.caad.fkie.fraunhofer.de/

Fig. 5. HTTP POST requests performed by the Astaroth malware family sample to exfiltrate data from an infected machine.

the figure, are used by the C&C for categorizing exfiltrated information from infected machines.

VII. Conclusion

In this paper, we proposed a detection method for passively discovering anomalies in the encrypted traffic of a monitored network. Relying on a combination of machine learning and statistical methods, the proposed solution identifies anomalous and rare SSL/TLS connections that exchange certificates deviating from the ones usually used in the monitored network. The method has been evaluated both on a malicious and a benign dataset: results show high accuracy in detecting malicious samples developed by worldwide leading threat actors and a very low false positive rate on legitimate encrypted traffic.

References

[1] C. Skipper. (2020, 09) The relevance of network security in an encrypted world. [Online]. Available: https://blogs.vmware.com/networkvirtualization/2020/09/network-security-encrypted.html/

[2] A. P. Felt, R. Barnes, A. King, C. Palmer, C. Bentzel, and P. Tabriz, "Measuring https adoption on the web," in *Proceedings of the 26th USENIX Conference on Security Symposium*, 2017, p. 1323–1338.

[3] Decipher. (2019, 06) Encryption, privacy in the internet trends report. [Online]. Available: https://duo.com/decipher/encryption-privacy-in-the-internet-trends-report

[4] N. Shah. (2020, 08) Keeping up with the performance demands of encrypted web traffic. [Online]. Available: https://www.fortinet.com/blog/industry-trends/keeping-up-with-performance-demands-of-encrypted-web-traffic

[5] Google. (2021, 08) Google transparency report: Https encryption on the web. [Online]. Available: https://transparencyreport.google.com/https/overview?hl=en

[6] Cisco. (2019) Cisco encrypted traffic analytics. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/enterprise-networks/enterprise-network-security/nb-09-encrytd-traf-anlytcs-wp-cte-en.pdf

[7] ENISA. (2020) Enisa threat landscape - malware. [Online]. Available: https://www.enisa.europa.eu/publications/malware/at_download/fullReport

[8] M. Korolov, "Cyber security review," *Treasury & Risk*, 2012.

[9] R. W. Taylor, E. J. Fritsch, and J. Liederbach, *Digital crime and digital terrorism*. Prentice Hall Press, 2014.

[10] T. Yadav and R. A. Mallari, "Technical aspects of cyber kill chain," *arXiv preprint arXiv:1606.03184*, 2016.

[11] VMware. (2021) Vmware nsx network detection and response. [Online]. Available: https://www.vmware.com/products/nsx-network-detection-response.html

[12] F. Callegati, W. Cerroni, and M. Ramilli, "Man-in-the-middle attack to the https protocol," *IEEE Security & Privacy*, vol. 7, pp. 78–81, 2009.

[13] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel, "Disclosure: detecting botnet command and control servers through large-scale netflow analysis," in *Proceedings of the 28th Annual Computer Security Applications Conference*, 2012, pp. 129–138.

[14] B. Anderson and D. McGrew, "Machine learning for encrypted malware traffic classification: accounting for noisy labels and non-stationarity," in *Proceedings of the 23rd ACM SIGKDD International Conference on knowledge discovery and data mining*, 2017, pp. 1723–1732.

[15] ——, "Identifying encrypted malware traffic with contextual flow data," in *Proceedings of the 2016 ACM workshop on artificial intelligence and security*, 2016, pp. 35–46.

[16] C. Liu, L. He, G. Xiong, Z. Cao, and Z. Li, "Fs-net: A flow sequence network for encrypted traffic classification," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019.

[17] B. Anderson, S. Paul, and D. McGrew, "Deciphering malware's use of tls (without decryption)," *Journal of Computer Virology and Hacking Techniques*, vol. 14, no. 3, pp. 195–211, 2018.

[18] A. S. Shekhawat, F. Di Troia, and M. Stamp, "Feature analysis of encrypted malicious traffic," *Expert Systems with Applications*, vol. 125, pp. 130–141, 2019.

[19] H. Shi, H. Li, D. Zhang, C. Cheng, and X. Cao, "An efficient feature generation approach based on deep learning and feature selection techniques for traffic classification," *Computer Networks*, vol. 132, pp. 81–98, 2018.

[20] M. Shen, M. Wei, L. Zhu, and M. Wang, "Classification of encrypted traffic with second-order markov chains and application attribute bi-grams," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 8, pp. 1830–1843, 2017.

[21] C. Kaufman, R. Perlman, and M. Speciner, *Network Security: Private Communication in a Public World, Second Edition*, 2nd ed. USA: Prentice Hall Press, 2002.

[22] International Telecommunication Union. (2021) X.509 : Public-key and attribute certificate frameworks. [Online]. Available: https://www.itu.int/rec/T-REC-X.509-201910-I/en

[23] L. Swersky, H. O. Marques, J. Sander, R. J. Campello, and A. Zimek, "On the evaluation of outlier detection and one-class classification methods," in *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 2016, pp. 1–10.

[24] G. Jenks, *The Data Model Concept in Statistical Mapping*, 1967.

[25] B. Jiang, "Head/tail breaks: A new classification scheme for data with a heavy-tailed distribution," *The Professional Geographer*, 2012.

[26] Internet Assigned Numbers Authority. (2019, 02) Transport layer security (tls) extensions. [Online]. Available: https://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xhtml#tls-extensiontype-values-1

[27] Internet Engineering Task Force. (2018, 08) Elliptic curve cryptography (ecc) cipher suites for transport layer security (tls) versions 1.2 and earlier. [Online]. Available: https://tools.ietf.org/html/rfc8422#page-11

[28] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. USA: Cambridge University Press, 2008.

[29] R. Neuhaus. (2011) Gibberish-detector. [Online]. Available: https://github.com/rrenaud/Gibberish-Detector

[30] S. S. Keerthi and C.-J. Lin, "Asymptotic behaviors of support vector machines with gaussian kernel," *Neural computation*, 2003.

[31] L. Oneto, *Model selection and error estimation in a nutshell*. Springer, 2020.

[32] G. Alberti. (2017, 09) 'plotjenks': R function for plotting univariate classification using jenks' natural break method.

[33] F. J. Richards, "A Flexible Growth Function for Empirical Use," *Journal of Experimental Botany*, vol. 10, no. 2, pp. 290–301, 06 1959. [Online]. Available: https://doi.org/10.1093/jxb/10.2.290

[34] ENISA. (2020) Enisa threat landscape 2020 - botnet. [Online]. Available: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2020-botnet/at_download/fullReport

[35] CrowdStrike. (2021) Crowdstrike global threat report 2021. [Online]. Available: https://go.crowdstrike.com/rs/281-OBQ-266/images/Report2021GTR.pdf

[36] Sophos. (2020, 02) Nearly a quarter of malware now communicates using tls. [Online]. Available: https://news.sophos.com/en-us/2020/02/18/nearly-a-quarter-of-malware-now-communicates-using-tls/

[37] FireEye. (2019, 12) Stages of a malware infection. [Online]. Available: https://community.fireeye.com/s/article/000002205

[38] Abuse.ch. (2021) Ja3 fingerprints. [Online]. Available: https://sslbl.abuse.ch/ja3-fingerprints/

[39] MITRE. (2021) Astaroth. [Online]. Available: https://attack.mitre.org/software/S0373/

[40] S. Saeli, F. Bisio, P. Lombardo, and D. Massa. (2020) Dns covert channel detection via behavioral analysis: a machine learning approach.

[41] F. Bisio, S. Saeli, P. Lombardo, D. Bernardi, A. Perotti, and D. Massa, "Real-time behavioral dga detection through machine learning," in *International Carnahan Conference on Security Technology (ICCST)*. IEEE, 2017, pp. 1–6.

Fast Flux Service Network Detection via Data Mining on Passive DNS Traffic*

Filippo Sobrero[1], Daniele Ucci[1], and Federica Bisio[1]

aizoOn Technology Consulting, Strada del Lionetto 6, 10146 Turin, Italy
[name].[surname]@aizoongroup.com

Abstract

In the last decade, the use of fast flux technique has become established as a common practice to organise botnets in Fast Flux Service Networks (FFSNs), which are platforms able to sustain illegal online services with very high availability. In this paper, we report on an effective fast flux detection algorithm based on the passive analysis of the Domain Name System (DNS) traffic of a corporate network. The proposed method is based on the near-real-time identification of different metrics that measure a wide range of fast flux key features; the metrics are combined via a simple but effective mathematical and data mining approach. The proposed solution has been evaluated in a one-month experiment, performed by collecting DNS traffic and injecting pcaps associated with different malware campaigns, that leverage FFSNs and cover a wide variety of attack scenarios. An in-depth analysis of a list of fast flux domains and the detection of malware campaigns backed up by a FFSN infrastructure confirmed the reliability of the metrics used in the proposed algorithm. All the fast flux domains were detected with a very low false positive rate and the resulting false positives have very low anomaly indicators; a comparison of performance indicators with similar works shows a remarkable improvement.

Keywords: automated security analysis, malware detection, network security, passive traffic analysis, botnet, malware campaign, fast flux.

1 Introduction

During the last few years, the number of cyberattacks with relevant financial impact and media coverage has been constantly growing. As a result, many companies and organizations have been reinforcing investment to protect their networks, with a resultant increase in the research on this topic [33].

Over the last two decades, botnets have represented one of the most prominent sources of threats on the internet: they are networks of compromised computers (popularly referred to as zombies or bots), which are controlled by a remote attacker (bot herder). Botnets provide the bot herder with massive resources (bandwidth, storage, processing power), allowing for the implementation of a wide range of malicious and illegal activities, like spam, distributed denial-of-service attacks, spreading of malware (such as ransomware, exploit kits, banking trojans, etc.) [3, 8, 9, 10, 15].

A common practice for bot herders is to organise their bots in Fast Flux Service Networks (FFSNs): some bots, chosen from a pool of controlled machines, are used as front-end proxies that relay data between a (possibly unaware) user and a protected hidden server. The technique behind these structures is the *fast flux*, i.e., the rapid and repeated changing of an internet host and/or name server resource record in a Domain Name System (DNS) zone, resulting

*An extended version of this paper has been published in Information Security Conference Proceedings in 2018. The current version has been reviewed by taking into account new references, using a new IP-dispersion metrics, and updating experimental evaluations with more recent samples.

in rapid changes of the IP addresses to which the domain resolves. FFSNs make the tracing and the recovery of all the infected components extremely difficult, thus allowing for a very high availability for illegal online services related to phishing, dumps stores, and distribution of ransomware, info stealers, and click fraud [12, 19, 23, 32, 34, 42].

FFSNs have been known to cybersecurity experts for more than one decade [15, 32] and, in the last years, it obtained a spotlight [4, 8, 10, 17, 31, 35]. Indeed, many security researchers have studied large botnets (e.g., Dark Cloud, also known as Zbot network, and SandiFlux), which make massive usage of fast flux, in order to detect their malicious activities [28, 12, 18]. Even if, in the very recent past, the academic interest about finding new methodologies for detecting FFSNs has died away, threat actors still extensively use this technique to make their infrastructure highly available [13, 27, 29].

The standard approach to FFSNs detection is via the so-called *active* DNS analysis, i.e., by actively querying some domains and by collecting and analysing the answers: this strategy has been widely explored and allows for extensive analyses of botnets [4, 15, 16, 17, 18, 19, 21, 23, 24]. Instead, the algorithm described in the present work relies on *passive* analysis of the DNS traffic of a single network: it detects the fast flux domains without interaction with the network traffic, thus making the algorithm completely transparent inside and outside the monitored network; in particular, it cannot be uncovered by the attackers, who often control the authoritative name servers responsible for responding to DNS queries about their fast flux domains [25]. The proposed detection approach has been evaluated on 30 days of DNS traffic, as described in Sect. 5. The performance is much higher compared to a state-of-the-art analogous method [34]. Moreover, the analysis was performed near-real-time: the average execution time of the algorithm was 25 seconds, while the average time between two subsequent runs of the algorithm was 3 minutes (see Sect. 5 for more details).

As an additional test of the proposed approach, we carried out a detailed analysis on IPs — collected via active DNS analysis — associated with a list of fast flux domains gathered from [6, 39, 30, 40] and reported, for space constraints, only the obtained results. This investigation along with the successful detection of recent malware campaigns confirmed the reliability of the metrics used in the fast flux detection method proposed herein.

The paper is structured as follows. In Sect. 2, we discuss the most relevant features of FFSNs, with an outline of related works. In Sect. 3, we briefly describe *aramis*, the monitoring platform that contains the fast flux detection method which is the focus of this paper and which is described thoroughly in Sect. 4. Section 5 comprises a detailed discussion of the experimental results of the test of the proposed algorithm, while Sect. 6 contains further investigations on the FFSNs underlying some fast flux domains. Finally, we summarize obtained results in Sect. 7.

2 Background and Related Work

One of the first works providing an overview of the fast flux attacks was the Honeynet project [32]. In order to explain hidden operations executed by botnets, authors gave examples of both single and double fast flux mechanisms: while the first rapidly changes the A records of domains, the latter frequently changes both the A records and the NS records of a domain. Interested readers can find a review and a classification of fast flux attacks in [42, 2].

Content Delivery Network (CDN) and Round-Robin DNS (RRDNS) are legitimate techniques which are used by large websites to distribute the load of incoming requests to several servers. The response to a DNS query is evaluated by an algorithm which chooses a pool of IPs from a large list of available servers whose number can be of the order of thousands. As a result, the behaviour in terms of DNS traffic is very similar to the one of a FFSN, and in-

deed CDNs and RRDNSs represent the typical false positives in fast flux detection algorithms [15, 19, 34, 41, 36].

A large number of approaches have been proposed to detect FFSNs and to distinguish them from legitimate CDNs and RRDNSs. Most of them rely on active DNS analysis, which allows for the collection of a large number of IPs associated with a domain, thus simplifying the FFSNs detection, but they require the resolutions of domains that may be associated with malicious activities [10, 15, 17, 19, 23]. These methods, despite being appropriate for a deep analysis of FFSNs, have relevant drawbacks in implementations oriented to the monitoring of corporate networks [25, 34].

Some FFSN detection methods based on passive DNS analysis have been proposed. Some of them analyse the DNS traffic of a whole Internet Service Provider (ISP), thus taking in input the DNS traffic generated by many different networks. Perdisci et al. [25], in particular, performed a large-scale passive analysis of DNS traffic. They extract some relevant features from the DNS traffic and classified the domains via a C4.5 decision tree classifier. Berger et al. [8] and Stevanovic et al. [35] proposed two other approaches to analyse the DNS traffic of an ISP. Both methods are based on a tool called DNSMap and classify the bipartite graphs formed by the collected fully qualified domain names and the associated IPs. The first method searches for generic malicious usage of DNS, while the latter focuses on FFSNs. More recently, Surjanto and Lin have successfully clustered DNS traffic of an ISP, distinguishing with high accuracy FFSNs from CDNs [36].

Soltanaghaei and Kharrazi [34] and Zang et al. [41] proposed techniques for passive DNS analysis of a network. While the former requires a history for each domain to be evaluated and achieved 94.44% detection rate and 0.001% false positive rate in their best experiment, the latter combines an online and offline approach to detect FFSNs with an accuracy greater than 96%. Our algorithm employs a similar approach, but, with a more careful choice of employed metrics and achieves better results, while performing a near-real-time analysis (see Sects. 4 and 5 for details).

3 aramis

The proposed fast flux detection technique is included in a commercially available network security monitoring platform called *aramis* (Aizoon Research for Advanced Malware Identification System) [1, 9]. This software automatically identifies different types of malware and attacks in near-real-time and its structure can be outlined in four phases:

1. Collection: sensors located in different nodes of the network gather data , preprocess them in real-time and send the results to a NoSQL database.

2. Enrichment: data are enriched in the NoSQL database using the information obtained from the aramis Cloud Service, which collects intelligence from various OSINT sources and from internally managed sources.

3. Analysis: stored data are processed by means of two types of analysis: (i) advanced cybersecurity analytics which highlight specific patterns of attacks, among which DGAs [9] and fast flux, and (ii) a machine learning engine which spots deviations from the usual behaviour of each node of the network.

4. Visualization: results are presented in ad hoc dashboards to show and highlight anomalies.

The cycle of the four phases restarts after a time Δt which slightly depends on the traffic flow analysed and amounts to 182 ± 36 seconds on the network described in Sect. 5. A time Δt of this magnitude is the best trade-off between the near-real-time requirement and the need of a large amount of data in order to have statistically significant results.

4 Detection Method

The aim of the proposed detection method is the near-real-time identification of malicious fast flux via the passive monitoring of the DNS traffic of a single network. To this end, we leverage a simple but effective method that combines a data mining approach with statistics and it is composed by three steps of analysis: (i) filtering phase in which queries known to be non-malicious (e.g., popular domains, known CDNs, local domains, etc.) are removed, (ii) Metrics identification step allows to compute some key indicators over the queries remaining after filtering, and (iii) anomaly identification phase in which identified metrics are used to detect malicious fast flux. The parameters of the model have been estimated over a *validation set* formed by 30-days of DNS traffic captured from the network described in Table 1, and by 15 pcaps associated with different malware campaigns that leverage FFSNs, collected from the public repository [7] and [5], spanning from 2015 to 2020.

Table 1: Validation-network description

	30-days total	**one-hour average**
N. of machines	261	-
N. of connections	80 M	111 k
N. of resolved A-type DNS queries	12 M	17 k
N. of unique resolved A-type DNS queries	381 k	527

4.1 Filters

The algorithm receives resolved DNS requests of type A (which return 32-bits IPv4 addresses, in accordance with [14]) collected near-real-time from the monitored network. The first step consists in the application of the filters: for a detailed description of filters applied to the retrieved queries, refer to the extended version of this paper [20].

4.2 Metrics Identification

The DNS requests that survive the filters 4.1 are integrated with the history of the previous 30 days, saved locally. This allows for a more accurate evaluation of the behaviour of the domains, however a reliable assessment is already possible when the first answer is received. Among the remaining domains there are many new emerging CDNs and in order to distinguish them from the FFSNs — which is the main challenge in malicious fast flux detection — we identified some key indicators. Some of these indicators can be already evaluated after a single query (we call them *static metrics*), while others need a certain history (*history-based metrics*). The information regarding Autonomous Systems (ASs) and public networks used in the following metrics are retrieved from [22].

4.2.1 Static Metrics

The metrics described in this section are evaluated over all the IPs collected.

Maximum Answer Length. A relevant metric for the detection of malicious fast flux is the number of IPs returned in a single A query. In particular, we consider the maximum m_{al} of such value: a malicious fast flux is believed to typically have a m_{al} larger than a legitimate fast flux [15, 42].

Cumulative Number of IPs. Malicious fast flux typically employ a larger number of IPs (n_{IP}) compared with CDNs, due to the lower reliability of each single node [34].

Cumulative Number of Public Networks. Since the botnet underlying a malicious fast flux contains infected machines which are typically distributed quite randomly in different networks, the same is expected to be true for the IPs retrieved by the related queries [15, 42]. For this reason a malicious fast flux typically has a number of public networks (n_{net}) larger than a legitimate CDN.

Cumulative Number of ASs. For the same reason described above, FFSNs typically have a number of ASs (n_{AS}) larger than legitimate CDNs.

Rescaled AS-Fraction. Metrics defined as

$$f'_{\text{AS}} = \theta(n_{\text{AS}} - n_0)\left[1 - e^{-\left(\frac{n_{\text{AS}}-n_0}{s}\right)^2}\right] \cdot \frac{n_{\text{AS}} - 1}{n_{\text{IP}}}, \tag{1}$$

that quantifies the degree to which the IPs are dispersed in different AS. This quantity takes values from $f'_{\text{AS}} = 0$ (when all the IPs are in the same AS) to $f'_{\text{AS}} \sim 1$ (when each IP is in a different AS and the number of IPs is large) and encode additional information about the typical scales associated with n_{AS} for CDNs and FFSNs, respectively. In Eq. 1, $\theta(\cdot)$ is the Heaviside step function (i.e., $\theta(x)$ is 1 for positive x and 0 otherwise), s is a scale representing the average number of ASs in a typical CDN and n_0 is a threshold for n_{AS} below which the behaviour is not suspicious from the viewpoint of the number of ASs, and the ratio between n_{AS} and n_{IP} is the not-yet scaled AS fraction metrics f_{AS}.

IP-Dispersion. The analysis of the distribution of the retrieved IPs is another way to understand to which degree the structure underlying FFSN is random and chaotic. We transform the set of the n IPs associated with each query into the corresponding positions in the 32-bits IPv4 address space $x_1, ...x_n$,[1] and we define (i) $\Delta\vec{x} = (x_i - x_{i-1})_{i=2}^{n}$ as the ordered sequence of transformed IPs so that $x_i \geq x_{i-1}$, (ii) l_n as the average distance as if the n IPs were uniformly distributed in the whole public IPs address space, and (iii) q as the $95th$ percentile. From $\Delta\vec{x}$, we remove elements $\Delta\vec{x}_i$ both bigger than l_n and above q, obtaining $\Delta\vec{y} = \{\Delta\vec{\tilde{x}}_j\}_{j=1}^{m}$. Indeed, values higher than l_n are more likely to correspond to distances between two distinct IP pools than to one between two subsequent contacted IPs, and such values are not very significant for the IP-dispersion metric. We exclude them along with the outliers of the distribution (i.e., $\Delta\vec{x}_i > q$) to focus on the effective distance among contacted IPs belonging to the same pool. The IP-dispersion is computed as follows

$$d_{\text{IP}} = \begin{cases} \frac{1}{l_n}\,\text{mean}(\Delta\vec{y}) & \text{if} \quad m \geq 0.75 \cdot (n-1), \\ \frac{1}{l_n}\,\text{median}(\Delta\vec{x}) & \text{otherwise} \end{cases} \tag{2}$$

where the original median metrics, proposed in [20], is applied only if the cardinality n of $\Delta\vec{x}$ is low[2]. Indeed, the mean better represents dispersion even in the presence of repetitive values

[1] To each IP $n_1.n_2.n_3.n_4$ we associated $x = 256^3\, n_1 + 256^2\, n_2 + 256\, n_3 + n_4$.

[2] The choice of which metrics the IP-dispersion should use is based upon experimental evaluations on the validation set.

of $\Delta\vec{x}$. The IP-dispersion takes value from $d_{\text{IP}} = 1$ (i.e., when the IPs are uniformely distributed among the whole public IP space) to $d_{\text{IP}} = 0$ (i.e., when the IPs are clearly subdivided into a few clusters of close addresses). A similar idea was used by Nazario et al. [23], who evaluated only the average distance among the $\{x_i\}$, but their metric is more sensitive to outliers and it is not normalised in the interval [0,1], which is crucial to combine it with the other metrics, as described in Sect. 4.3. The FFSNs analysis described in Sect. 6 confirmed that the indicator in Eq. 2 is able to catch the key distribution properties of IPs in a FFSN.

4.2.2 History-Based Metrics

The *history* is constructed by subdividing the queries retrieved from the monitored network in subsequent *chunks*: each chunk contains at least 10 queries and spans a time interval of at least one hour; these two conditions are the minimal requirements to make the metric definitions meaningful from a statistical point of view. The metrics described in this section are evaluated only if it is possible to construct at least two chunks.

Change in the set of IPs. It is a common belief that, while a CDN typically returns IPs taken from a stable IP-pool, a malicious fast flux employs the available nodes in the FFSN, which often evolves quickly, and therefore its IP-pool changes from time to time [15, 42]. We defined a metric which measures in a very simple way the change in the IP-pool $c_{\text{IP}} = n_{\text{IP}}/n^{\text{c}}_{\text{IP}} - 1$, where n^{c}_{IP} is the number of unique IPs present in the chunk averaged over all chunks, while n_{IP} has been defined in Sect. 4.2. This quantity takes the value $c_{\text{IP}} = 0$ when all the IPs are found in each chunk, i.e., when the IP-pool is stable and it is explored completely in each chunk (and therefore $n^{\text{c}}_{\text{IP}} = n_{\text{IP}}$). On the other hand, if the IP-pool changes substantially from one chunk to the other, the total number of IPs n_{IP} is much larger than the average number of IPs n^{c}_{IP} found in a chunk, and therefore c_{IP} becomes large (it is unbounded above). The same considerations apply to all the following metrics.

Change in the Set of Public Networks. While CDNs typically use IPs taken from the same few public networks, malicious fast flux frequently introduce IPs from new networks [15, 42]. We measure the change in the set of public networks by means of $c_{\text{net}} = n_{\text{net}}/n^{\text{c}}_{\text{net}} - 1$, where $n^{\text{c}}_{\text{net}}$ is the network-analogous of n^{c}_{IP}.

Change in the Set of ASs. The generalisation of the previous argument to the next aggregation level brings us to the analysis of the changes in the number of AS involved. We introduce therefore $c_{\text{AS}} = n_{\text{AS}}/n^{\text{c}}_{\text{AS}} - 1$, where n^{c}_{AS} is the AS-analogous of n^{c}_{IP}.

Change in the Answer Length. Another relevant indicator is the change in the number of IPs retrieved in each query [15, 42]. We measure this change by means of $c_{\text{al}} = m_{\text{al}}/m^{\text{c}}_{\text{al}} - 1$, where m^{c}_{al} is the m_{al}-analogous of n^{c}_{IP}.

4.3 Fast Flux Domains Identification

A preliminary step for fast flux domains identification is the filtering of the queries with $d_{\text{IP}} = 0$, because this removes many false positives with no loss in terms of true positives. The next step is the use of the metrics defined in Sect. 4.2 to discriminate among malicious fast flux and CDN. We aggregate the static and history-based metrics separately, and finally we combine them into a single anomaly indicator A, which can straightforwardly be used to classify the queries between fast flux and legit domains.

4.3.1 Aggregation of the Static Metrics

We normalised the metrics n_{IP}, n_{net}, n_{AS}, and m_{al} in the interval [0,1], so that for all of them the value 0 corresponds to a typical CDN, while 1 corresponds to the expected behaviour of a malicious fast flux. This is achieved by means of a square-exponential scaling of the form

$$x \longrightarrow 1 - e^{-\left(\frac{x-x_0}{s}\right)^2}, \tag{3}$$

where $x_0 = 1$ is the minimum value for the metric before the rescaling, s is different for each metric and represents an intermediate scale between a typical CDN behaviour and a behaviour clearly ascribed to a malicious fast flux.[3] After the scaling, we combined these indicators with a weighted arithmetic mean in a unique static index[4]

$$A_{\text{stat}} = w_{\text{IP}} n_{\text{IP}} + w_{\text{net}} n_{\text{net}} + w_{\text{AS}} n_{\text{AS}} + w_{\text{al}} m_{\text{al}} + w_f f_{\text{AS}} + w_d d_{\text{IP}}. \tag{4}$$

4.3.2 Aggregation of the History-Based Metrics

As already mentioned, the metrics c_{IP}, c_{net}, c_{AS}, and c_{al} defined in Sect. 4.2 are unbounded above. We normalise them in the interval [0,1] by means of Eq. 3 with $x_0 = 0$ (as the minimum value for these metrics before the rescaling is 0).[5] After the rescaling, all the metrics take values in the interval [0,1] and for each of them a value close to 0 corresponds to a very stable behaviour, while a value close to 1 indicates a behaviour with high variability over time. We combine then in a unique indicator three of the history-based metrics (the fourth, i.e., c_{al} is instead used in Eq. 6) with a weighted arithmetic mean[6]

$$A_{\text{dyn}} = w'_{\text{IP}} c_{\text{IP}} + w'_{\text{net}} c_{\text{net}} + w'_{\text{AS}} c_{\text{AS}}. \tag{5}$$

4.3.3 Final Aggregation

We combine the indicators A_{stat}, A_{dyn}, and c_{al} into a single anomaly indicator A, which should be used to classify the queries between fast flux and legit domains. In order to reduce false positives, we differentiate on the basis of the quantity f_{AS}, and we define

$$A = \begin{cases} \sum_i w_i A_i & \text{if} \quad f_{\text{AS}} \geq 0.5 \\ \prod_i (A_i)^{w_i} & \text{if} \quad f_{\text{AS}} < 0.5 \end{cases}, \tag{6}$$

Table 2: Test-network description

	30-days total	**one-hour average**
N. of machines	391	-
N. of client machines	286	-
N. of connections	70 M	100 k
N. of resolved A-type DNS queries	27 M	40 k
N. of unique resolved A-type DNS queries	200 k	820

[3] The values of s were set based on information retrieved from the literature ([42] and references therein) and the validation set. More in detail, we chose $s_{\text{IP}} = 24$, $s_{\text{net}} = 12$, $s_{\text{AS}} = 6$, and $s_{\text{al}} = 10$.

[4] The weights reflect the importance of the corresponding metric in the correct classification in the validation set; the optimal values are $w_{\text{IP}} = w_{\text{net}} = 0.03$, $w_{\text{AS}} = 0.13$, $w_{\text{al}} = 0.09$, $w_f = 0.54$, and $w_d = 0.18$.

[5] The values of s were set based on information retrieved from the literature and the validation set. More in detail, we chose $s_{\text{IP}} = s_{\text{net}} = 1$ and $s_{\text{AS}} = s_{\text{al}} = 0.5$.

[6] The weights reflect the importance of the corresponding metric in the validation set; the optimal values are $w'_{\text{IP}} = 0.07$, $w'_{\text{net}} = 0.23$, and $w'_{\text{AS}} = 0.7$.

where $\{A_i\} = \{A_{\text{stat}}, A_{\text{dyn}}, c_{\text{al}}\}$ and $\{w_i\}$ are the related weights.[7]

The detection of malicious fast flux has thus been reduced to a very simple one-dimension classification problem: only queries with $A > A_{\text{th}}$ are labeled as fast flux, where the optimal threshold ($A_{\text{th}} = 0.25$) has been found by maximizing the performance on the validation set. In order to increase the readability of the results, we applied a sigmoid-shaped rescaling which maps $A = 0$ and $A = 1$ onto themselves and A_{th} onto 0.5.

5 Experimental Evaluation

The fast flux detection algorithm described in Sect. 4 was evaluated over a test set comprising 30 days of ordinary traffic of the network described in Table 2 with the injection of fast flux traffic which covers all the most relevant fast flux attack scenarios (see Table 3 for a complete list). Note that the test set has been only used to test the performance of the algorithm and not to modify the algorithm or the parameters. In addition, the test set has been updated to 2021 by collecting again the dataset and its network-related information.

Table 3: Malware description

Category	**Campaign**	**Domains (A)**	$\langle A \rangle$
Banking Trojan	ZBOT	miscapoerasun.ws (0.85)	0.85
Banking Trojan	Dreambot	rahmatulahh.at (0.89); ardshinbank.at (0.92)	0.91
Banking Trojan	Ursnif	widmwdndghdk.com (0.90); bnvmcnjghkeht.com (0.85); qqweerr.com (0.85)	0.87
VBA Dropper	Doc Dropper Agent[a]	aassmcncnnc.com (0.90); iiieeejrjrjr.com (0.87); ghmchdkenee.com (0.88)	0.88
Ransomware	Locky	thedarkpvp.net (0.83); nsaflow.info (0.91); mrscrowe.net (0.93); sherylbro.net (0.87); gdiscoun.org (0.90); scottfranch.org (0.90)	0.89
Ransomware	Nymaim	iqbppddvjq.com (0.91); danrnysvp.com (0.91); pmjpdwys.com (0.93); vqmfcxo.com (0.86); gbfeiseis.com (0.91); danrnysvp.com (0.87); iuzngzhl.com (0.97); vpvqskazjvco.com (0.84); jauudedqnm.com (0.93); dtybgsb.com (0.93); tuzhohg.com(0.93); sxrhysqdpx.com (0.86); arlfbqcc.com (0.93)	0.90
Banking Trojan	Zeus Panda	farvictor.co (0.89); fardunkan.co (0.89); bozem.co (0.84); farmacyan.co (0.87); fargugo.co (0.90); manfam.co (0.85)	0.87
Banking Trojan	GOZI ISFB	qdkngijbqnwehiqwrbzudwe.com (0.80); jnossidjfnweqrfew.com (0.90); zxciuniqhweizsds.com (0.86); huwikacjajsneqwe.com (0.92); efoijowufjaowudawd.com (0.92); onlyplacesattributionthe.net (0.90); nvvnfjvnfjcdnj.net (0.86); popoiuiuntnt.net (0.89); zzzzmmmsnsns.net (0.80); popooosneneee.net (0.83); liceindividualshall.net (0.87); roborobonsnsnn.net (0.93)	0.87
Ransomware	GandCrab	zonealarm.bit (0.90)	0.90
Backdoor	ServHelper	medastr.com (0.74)	0.74
Banking Trojan	Ursnif	goose-mongoose.at (0.84); roiboypo.ru (0.84); roiboypoleno.ru (0.84); thatallmafaka.at(0.78); to4karu.ru(0.78); zvednyisvet.ru(0.78)	0.81

[a] Doc.Dropper.Agent-6332127-0 [37]

[7] An optimisation procedure on the validation set produced similar weight for the three quantities: $w_{\text{stat}} = 0.27$, $w_{\text{dyn}} = 0.38$, and $w_{\text{al}} = 0.35$.

The fast flux traffic has been injected in the network via 50 pcaps — collected from public repositories [6, 39, 30, 5] — which are associated with 11 different malware campaigns, widely spread in the very last years. It is worth noting that, with respect to the extended version of this paper [20], we have included 3 new samples (listed in the last two rows of Table 3) to show how fast flux are still a relevant attack technique. In particular, Table 3 provides a brief description of each malware campaign with the following information: the category (i.e., the malware type associated with the campaign), the name of the campaign, the list of the domains present in each pcap of the campaign with the anomaly indicator A associated by the algorithm to each one of them, and the average value of A for each campaign.

Table 3 clearly shows that the proposed method successfully detected all the fast flux domains with a high anomaly indicator. In fact, the value of A averaged over all campaigns is equal to 0.86.

In Table 4, we summarise the performance of the algorithm: in the second column we consider the total number of outputs of the algorithm (i.e., the number of domains with $A > 0$) while in the third column we report the number of outputs labeled as fast flux (i.e., the number of domains with $A > 0.5$). On the rows, we report the following quantities: the number of unique fast flux domains correctly detected (i.e., True Positives T_P), the ones incorrectly labeled as legit (i.e., False Negatives F_N), the number of unique legit domains incorrectly labeled as fast flux (i.e., False Positives F_P), and their corresponding rates. All rates are given as absolute values and as percentages for each type on the corresponding number of unique domains in input.

It is important to note that, with the application of the new metric reported in Equation 2, the algorithm generates more output events but, in the majority of the cases, the anomaly score A is less than 0.5, making the actual number of queries wrongly labeled as fast flux lower. Another remarkable result is the absence of false negatives: this determines indeed a 100% recall, also known as detection rate, $R = T_P/(T_P + F_N)$. In order to evaluate the algorithm also with a metric that takes into account the false positives rate F_P, we computed the F-score $F = 2\,P\,R/(P + R)$ (where $P = T_P/(T_P + F_P)$), obtaining $F = 99.0\%$.

As a comparison, [34] obtained $R = 94.4\%$ and $F = 89.5\%$ in their best experimental result. We can therefore conclude that the proposed method is able to detect queries to fast flux domains in a corporate network in near-real-time and with high anomaly indicators, limiting false positives at the same time.

Table 4: Results

	$A > 0$	$A > 0.5$
True Positives (T_P)	54 (100%)	54 (100%)
False Negatives (F_N)	0 (0%)	0 (0%)
False Positives (F_P)	13 (<0.065%)	1 (<0.001%)

6 Fast Flux Service Networks Detection in aramis

The extended version of this paper [20] reports an in-depth analysis of the proposed FFSN detection method on malicious IPs, collected for over a month in 2018 via active DNS analysis, from a set of public repositories [6, 39, 30, 40]. Such analysis revealed that a large number of IPs corresponds to two famous FFSNs: Dark Cloud and SandiFlux, responsible for widespread malware campaigns [26, 38, 12, 18, 28]. Both FFSNs have been correctly identified by our

approach and, as we mentioned in Section 3, the proposed detection method has been included in our commercially available network security monitoring platform *aramis*.

In 2019 and 2020, *aramis* has been able to detect many attacks relying on fast-flux techniques as, for example, one of the offspring of the original Gozi, named Ursnif [38, 11]. It is a widely distributed banking Trojan that uses Microsoft Office Documents to get into victims' machines and, then, contact its command-and-control server to further receive additional commands. After a thorough investigation, we have discovered that the attacks are part of a greater malicious spam campaign targeting companies through fraudulently branded e-mails, familiar to the recipient victims. More recently, *aramis* have also detected an unidentified adware that relies on a FFSN. Table 5 shows a summary of some of the metrics presented in Section 4 and evaluated over Dark Cloud and Sandiflux, the into-the-wild attacks detected by *aramis*, and three large CDNs. It is worth noting that even if Ursnif and the unidentified adware do not have history-based associated features, and two of the reported CDNs (i.e., www.nationalgeographic.it and cdn.wetransfer.net) have a very large number of IPs, the proposed detection method can still identify FFSNs.

Table 5: Summary of some relevant metrics

	n_{IP}	n_{AS}	$n_{\text{IP}}^{\text{resc}}$	$n_{\text{AS}}^{\text{resc}}$	$c_{\text{AS}}^{\text{resc}}$	$f_{\text{AS}}^{\text{resc}}$	d_{IP}
Dark Cloud	2856	354	1	1	1	1	$1.2\ 10^{-3}$
Sandiflux	6203	1517	1	1	1	1	0.69
Ursnif	10	10	0.13	0.90	-	0.90	0.18
Unidentified adware	10	1	1	0	-	0	0.33
www.nationalgeographic.it	2478	1	1	0	0	0	$2.6\ 10^{-6}$
cdn.wetransfer.net	2734	1	1	0	0	0	$2.9\ 10^{-6}$
neo4j.com	29	1	0.74	0	0	0	$5.1\ 10^{-4}$

7 Conclusions

In this paper, we proposed a fast flux detection method based on the passive analysis of the DNS traffic of a corporate network. The analysis is based on *aramis* security monitoring system. The proposed solution has been evaluated by injecting 50 pcaps associated with 11 different malware campaigns that leverage FFSNs and cover a wide variety of attack scenarios, ranging from 2015 to 2020. The improvement of the IP-dispersion metrics allowed *aramis* not only to correctly detect all the fast flux with a very low false positive rate, but also to reduce the false positives that have an anomaly indicator higher than 50%. The comparison of performance indicators with a state-of-the-art work shows a remarkable improvement. An in-depth active analysis of a list of malicious fast flux domains and the detection of malware campaigns backed up by a FFSN infrastructure confirmed the reliability of the metrics used in the proposed algorithm.

Acknowledgments

We would like to thank the aizoOn I-SOC, with special mention to Matteo Zorzino, for the helpful comments and references to investigations, carried out internally by the I-SOC, on occurred security incidents. Both aspects allowed to improve the overall quality of this paper.

References

[1] aizoOn. https://www.aramisec.com.

[2] Ahmad Al-Nawasrah, Ammar Ali Almomani, Samer Atawneh, and Mohammad Alauthman. A survey of fast flux botnet detection with fast flux cloud computing. *International Journal of Cloud Applications and Computing (IJCAC)*, 10(3):17–53, 2020.

[3] Kamal Alieyan, Ammar Almomani, Ahmad Manasrah, and Mohammed M Kadhum. A survey of botnet detection based on dns. *Neural Computing and Applications*, 28(7):1541–1558, 2017.

[4] Ammar Almomani. Fast-flux hunter: a system for filtering online fast-flux botnet. *Neural Computing and Applications*, 29(7):483–493, 2018.

[5] Any Run: Interactive Malware Analysis. https://app.any.run/.

[6] Hybrid Analysis. https://www.hybrid-analysis.com/.

[7] Malware Traffic Analysis. https://www.malware-traffic-analysis.net/.

[8] Andreas Berger, Alessandro D'Alconzo, Wilfried N Gansterer, and Antonio Pescapé. Mining agile dns traffic using graph analysis for cybercrime detection. *Computer Networks*, 100:28–44, 2016.

[9] Federica Bisio, Salvatore Saeli, Pierangelo Lombardo, Davide Bernardi, Alan Perotti, and Danilo Massa. Real-time behavioral dga detection through machine learning. In *Security Technology (ICCST), 2017 International Carnahan Conference on*, pages 1–6. IEEE, 2017.

[10] Prabhjot Singh Chahal and Surinder Singh Khurana. Tempr: application of stricture dependent intelligent classifier for fast flux domain detection. *International Journal of Computer Network and Information Security*, 8(10):37, 2016.

[11] Yoroi Company. Ursnif: The latest evolution of the most popular banking malware. https://yoroi.company/research/ursnif-the-latest-evolution-of-the-most-popular-banking-malware/.

[12] Wayne Crowder and Noah Dunker. Dark cloud network facilitates crimeware.

[13] Yael Daihes. Detecting mylobot, unseen dga based malware, using deep learning. https://blogs.akamai.com/sitr/2021/01/detecting-mylobot-unseen-dga-based-malware-using-deep-learning.html.

[14] Internet Engineering Task Force. https://tools.ietf.org/html/rfc1035.

[15] Thorsten Holz, Christian Gorecki, Konrad Rieck, and Felix C Freiling. Measuring and detecting fast-flux service networks. In *NDSS*, 2008.

[16] Ching-Hsiang Hsu, Chun-Ying Huang, and Kuan-Ta Chen. Fast-flux bot detection in real time. In *International Workshop on Recent Advances in Intrusion Detection*, pages 464–483. Springer, 2010.

[17] Ci-Bin Jiang and Jung-Shian Li. Exploring global ip-usage patterns in fast-flux service networks. *JCP*, 12(4):371–379, 2017.

[18] Or Katz, Raviv Perets, and Guy Matzliach. Digging deeper – an in-depth analysis of a fast flux network, 2017.

[19] Hui-Tang Lin, Ying-You Lin, and Jui-Wei Chiang. Genetic-based real-time fast-flux service networks detection. *Computer Networks*, 57(2):501–513, 2013.

[20] Pierangelo Lombardo, Salvatore Saeli, Federica Bisio, Davide Bernardi, and Danilo Massa. *Fast Flux Service Network Detection via Data Mining on Passive DNS Traffic: 21st International Conference, ISC 2018, Guildford, UK, September 9–12, 2018, Proceedings*, pages 463–480. Springer, 01 2018.

[21] Sergi Martinez-Bea, Sergio Castillo-Perez, and Joaquin Garcia-Alfaro. Real-time malicious fast-flux detection using dns and bot related features. In *Privacy, Security and Trust (PST), 2013 Eleventh Annual International Conference on*, pages 369–372. IEEE, 2013.

[22] MaxMind. https://dev.maxmind.com/geoip/.

[23] Jose Nazario and Thorsten Holz. As the net churns: Fast-flux botnet observations. In *Malicious*

and Unwanted Software, 2008. MALWARE 2008. 3rd International Conference on, pages 24–31. IEEE, 2008.

[24] Emanuele Passerini, Roberto Paleari, Lorenzo Martignoni, and Danilo Bruschi. Fluxor: Detecting and monitoring fast-flux service networks. In *International conference on detection of intrusions and malware, and vulnerability assessment*, pages 186–206. Springer, 2008.

[25] Roberto Perdisci, Igino Corona, and Giorgio Giacinto. Early detection of malicious flux networks via large-scale passive dns traffic analysis. *IEEE Transactions on Dependable and Secure Computing*, 9(5):714–726, 2012.

[26] Pnwcode. http://www.pwncode.club/2017/09/dreambot-targeting-bulgarian-users.html.

[27] Check Point. February 2020's most wanted malware: Increase in exploits spreading the mirai botnet to iot devices. https://blog.checkpoint.com/2020/03/11/february-2020s-most-wanted-malware-increase-in-exploits-spreading-the-mirai-botnet-to-iot-devices/.

[28] Proofpoint. https://www.proofpoint.com/us/threat-insight/post/sandiflux-another-fast-flux-infrastructure-used-malware-distribution-emerges.

[29] Proofpoint. A comprehensive look at emotet's summer 2020 return. https://www.proofpoint.com/us/blog/threat-insight/comprehensive-look-emotets-summer-2020-return.

[30] Reverse. https://www.reverse.it/.

[31] Jukka Ruohonen and Ville Leppänen. Investigating the agility bias in dns graph mining. In *Computer and Information Technology (CIT), 2017 IEEE International Conference on*, pages 253–260. IEEE, 2017.

[32] William Salusky and Robert Danford. Know your enemy: Fast-flux service networks. *The Honeynet Project*, pages 1–24, 2007.

[33] Australian Computer Society. https://www.acs.org.au/content/dam/acs/acs-publications/ACS_Cybersecurity_Guide.pdf.

[34] Elaheh Soltanaghaei and Mehdi Kharrazi. Detection of fast-flux botnets through dns traffic analysis. *Scientia Iranica. Transaction D, Computer Science & Engineering, Electrical*, 22(6):2389, 2015.

[35] Matija Stevanovic, Jens Myrup Pedersen, Alessandro D'Alconzo, and Stefan Ruehrup. A method for identifying compromised clients based on dns traffic analysis. *International Journal of Information Security*, 16(2):115–132, 2017.

[36] Williams Surjanto and Charles Lim. Finding fast flux traffic in dns haystack. In Awais Rashid and Peter Popov, editors, *Critical Information Infrastructures Security*, pages 69–82, Cham, 2020. Springer International Publishing.

[37] Cisco Talos. http://blog.talosintelligence.com/2017/07/threat-roundup-0630-0707.html.

[38] Cisco Talos. http://blog.talosintelligence.com/2018/03/gozi-isfb-remains-active-in-2018.html.

[39] Packet Total. https://packettotal.com/.

[40] Virus Total. https://virustotal.com/.

[41] Xiao-Dong Zang, Jian Gong, Shao-Huang Mo, Ahmad Jakalan, and De-Lin Ding. Identifying fast-flux botnet with agd names at the upper dns hierarchy. *IEEE Access*, 6:69713–69727, 2018.

[42] Shijie Zhou. A survey on fast-flux attacks. *Information Security Journal: A Global Perspective*, 24(4-6):79–97, 2015.

Towards an Automated Pipeline for Detecting and Classifying Malware through Machine Learning*

Nicola Loi[1], Claudio Borile[2], and Daniele Ucci[2]

[1] Università degli Studi di Torino, via Pietro Giuria 1, 10125 Turin, Italy
[name].[surname]@edu.unito.it
[2] aizoOn Technology Consulting, Strada del Lionetto 6, 10146 Turin, Italy
[name].[surname]@aizoongroup.com

Abstract

The constant growth in the number of malware - software or code fragment potentially harmful for computers and information networks - and the use of sophisticated evasion and obfuscation techniques have seriously hindered classic signature-based approaches. On the other hand, malware detection systems based on machine learning techniques started offering a promising alternative to standard approaches, drastically reducing analysis time and turning out to be more robust against evasion and obfuscation techniques. In this paper, we propose a malware taxonomic classification pipeline able to classify Windows Portable Executable files (PEs). Given an input PE sample, it is first classified as either malicious or benign. If malicious, the pipeline further analyzes it in order to establish its threat type, family, and behavior(s). We tested the proposed pipeline on the open source dataset EMBER, containing approximately 1 million PE samples, analyzed through static analysis. Obtained malware detection results are comparable to other academic works in the current state of art and, in addition, we provide an in-depth classification of malicious samples. Models used in the pipeline provides interpretable results which can help security analysts in better understanding decisions taken by the automated pipeline.

Keywords: automated security analysis, malware pipeline, malware classification, malware detection, static analysis.

1 Introduction

Malware (short for malicious software) is the generic term used to refer to unwanted software developed to infect and interfere with the operations of a single machine or networks of computers [1, 2]. Since the first documented virus appeared in the 1970s, the evolution of computer science has always been accompanied closely by the creation of new, better and more harmful malicious software, in a constant fight between malware developers and security analysts. In recent years, though, the refinement and emergence of new software technologies have allowed an exponential growth in the number of malware in circulation, not only vertically (volume) but also horizontally (types and functionality) [3]. Together with the ever more sophisticated evasion techniques being developed by attackers, security experts and anti-malware vendors struggle to keep up the race by means of "standard" methods, i.e. signature-based and heuristics [4, 5, 6]. In this context, machine learning (ML) seems to be the most promising tool for an automated analysis and prevention of this kind of threats [7]. The strength of ML is its ability to automatically identify hidden patterns and correlations in large volumes of raw data, and exploit these statistical features to, in the case of malware analysis, recognise previously unseen

*This is a preliminary work on the development of a pipeline capable of exhaustively characterizing analyzed Windows PE samples.

attacks. Generally speaking, classic ML approaches for cyber security purposes focus on a first phase of features extraction through static, dynamic or hybrid analysis. These features are then used to train models that allow to classify malicious and benign files. More recently, the advancements in deep learning methods has inspired a series of studies exploiting other input formats, like raw binaries [8, 9] and image representations [10, 11, 12] among others. Historically, researchers and security vendors have mostly been more focused on creating models for the detection of malicious and benign files rather than exploring the possibility of using ML for an in-depth analysis of single malware samples. A reason for this might be the difficulty in collecting large and well-annotated datasets, a complex and expensive task, together with the intrinsic difficulty in classifying the characteristics of malwares due to the presence of many variants and the lack of a standard nomenclature [13, 14, 15].

Endgame — now Elastic — released in 2017 the first version of an open-source dataset called EMBER (Elastic Malware Benchmark for Empowering Researchers) containing semi-raw static features from more than 1 million Portable Executable (PE). In 2018, they released a second version of the dataset, using more recent malware, correcting issues in the data collection, and providing also labels for malware classes, using the open source tool AVClass [16]. This tool allows to parse and organise, in a definite taxonomy, the different nomenclatures returned by multiple anti-virus vendors. Exploiting and integrating the last version of the EMBER dataset as described in detail below, we explore the possibility of training a machine learning pipeline for a full, automated analysis of PEs using static features, from malware detection to classification by threat-type, family, and behaviour. At the time of writing, there is no academic paper that has explored the possibility of leveraging the EMBER dataset for a taxonomic malware classification. The paper is organized as follows: Section 2 presents related works, while Section 3 details the EMBER dataset. The proposed classification pipeline is described in Section 4 and experimental evaluation results are reported in 5.

2 Related Work

In the last decade, the number of studies on machine learning detection techniques is constantly increased thanks to: i) the recent growth of new and powerful algorithms and data wrangling methods, ii) the increase in computational capabilities, and iii) the availability of public, annotated malware datasets [17, 18, 19, 20, 21]. However, most of these works only consider the problem of distinguishing malicious software from benign. Furthermore, many papers rely on small or outdated datasets that are unlikely to provide a statistically-significant representation of malware population and labelling procedures create a bias towards easier datasets [22, 23]. The above considerations resulted in a general difficulty in assessing the real performance of these methods "into the wild" and a general incapacity to deliver models that can be effectively deployed, despite notable results summarized in [22]. Nevertheless, many recent works have been very effective in detecting malware, reaching almost perfect performances in accuracy.

In this scenario, the authors of the EMBER dataset provided a benchmark model in 2018, trained on their latest release of the model, obtaining a 86.8% detection rate at 0.1% False Positive Rate (FPR). Results obtained in the previous model release reported a 93% detection rate [24], meaning that the EMBER dataset developers have successfully hardened the process of correctly classifying malicious samples. Despite more recent works on the same dataset provide small improvements in the detection rate [25, 26], they usually rely on deep learning frameworks that make more difficult to interpret model outputs.

To our knowledge, there are no published works that used the 2018 EMBER dataset for a multi-class classification in malware families. Among the studies that tackle the problem

of classifying malware families [27, 28, 29, 30], Ahmadi *et al.* [10] considered the Microsoft Malware Classification Challenge data, a labeled dataset of about 20,000 malware samples representing a mix of 9 different families[1], to build a model for malware families classification. While the dataset is much smaller than EMBER and thus the performances are difficult to compare, they obtained a notable 0.997 accuracy over a 5-fold cross validation, considering a set of features similar to those used in the EMBER model, suggesting their robustness for both malware detection and family classification.

Malware behaviour classification is usually based on dynamic or hybrid analysis [31, 32, 33]. This is not surprising, since malware behaviour is expected to manifest itself only upon execution [22]. Nevertheless, we believe that it is still interesting to assess behavioral classification on purely static analyses contained in EMBER. We have not find any previous work considering the problem of an integrated, hierarchical malware classification into threat-type, family, and behaviour using features from static analysis.

3 Data description

3.1 The EMBER dataset

In this work we consider the 2018 release of the EMBER (Elastic Malware Benchmark for Empowering Researchers) dataset, an open source benchmark collection of 1 million PEs scanned in or before 2018 [34]. The data comes split in two separate sets containing the training and test data. The training set consists of 600,000 labeled samples (benign or malware) and 200,000 unlabeled samples the we did not consider in this study. The test set contains 200,000 labeled samples. The authors claim that the particular splitting between train and test is specifically engineered for having a "harder" dataset with respect to the first release [24]. Each sample comes as a JSON object and is uniquely identified via its sha256 and md5 hashes, and provides semi-raw information for static malware analysis parsed using the LIEF open source package [35] and divided in nine major groups: General, header, and section information, imported and exported functions, strings information, raw-byte histogram, and byte-entropy histogram. Finally, additional information is given on the label (0 for benign files, 1 for malicious files, and -1 for unlabeled files), the coarse time stamp of estimate first detection of the malware, the malware class extracted from the VirusTotal [36] report using the open source tool avclass [16]. A full description of the dataset can be found in [34, 24]. As explained in detail in section 4, we consider all the 600,000 labeled training data for training and validating each stage of the classification pipeline, while the test set will be used only to asses the global performance of the pipeline, mimicking its usage in a real-world scenario.

3.2 Threat-type and behaviour labels collection

The EMBER dataset only provides the malware/benign and a generic "avclass" labels (the output of the AVClass tool [16] for family tagging). In order to classify a specific malware sample by threat-type, family, and behavior, we used the open-source tool AVClass2 [37], not yet available when the EMBER dataset came out, to systematically extract information on the malware threat-type and behavior for each labeled sample in the dataset. Our final dataset consists of the data from the EMBER original dataset plus four labels: malware/not-malware, and if malware its threat-type, family and behavior classes where available. It is important to note that AVClass2 parses the malware label returned from each of the AV vendors and returns

[1] https://www.kaggle.com/c/malware-classification

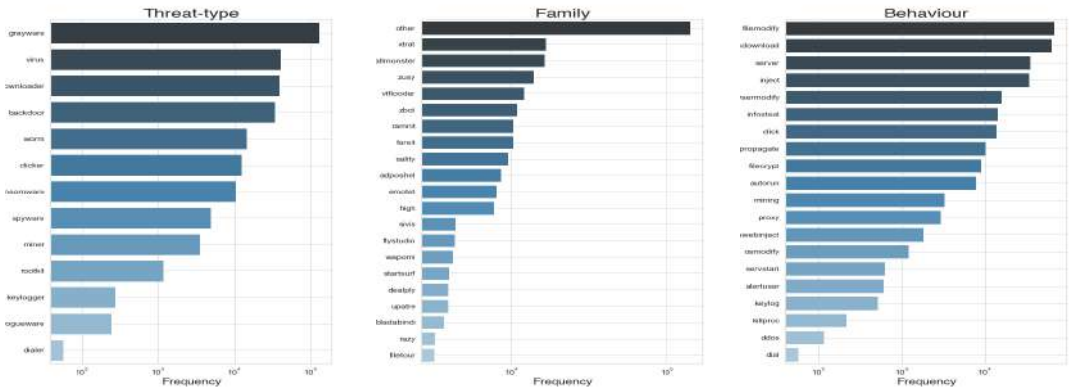


Figure 1: Frequency of the target classes for the three different types of classification stages. The horizontal axis is in logarithmic scale.

a ranking of the returned labels, not necessarily a single result per class. As a consequence, while a powerful tool tailored for each AV vendor and including a complete malware taxonomy, still suffers from the lack of a shared nomenclature across different security vendors. Furthermore, the estimated accuracy of the output labels is reported to be around 90% for families (threat-type and behavior not reported) [37], so that we must take into consideration the inherent imperfect labelling of the data.

3.3 Classification target

Looking at figure 1 we can see that for each of the classification stages of the pipeline the classes are heavily unbalanced (note the logarithmic scale). In particular, the "Family" group has a very large number of poorly populated classes in the train set. We thus kept the first 20 families and grouped all the remaining in a single class named "other", that now represents the most populous class. We will discuss the effects of this unbalance in section 5. For the threat-type we can see that, as expected, the most represented classes are greyware, viruses, and downloaders, while dialers are by far the less common with only few tens of samples. Regarding malware behavior, the first two classes -filemodify and execdownload- make half of the total samples. Coherently with the threats, class "dial" is the class with fewest samples.

4 Pipeline Architecture Overview

As introduced in Section 1, the main goal of the proposed pipeline is to accurately classify malware using different stages of processing, each one leveraging a properly trained classifier. Figure 4 shows how we implemented the pipeline: the first stage detects whether input samples are malicious or benign; those classified as malicious are propagated to the second stage, which is responsible for labeling them with a known malware category (e.g., virus, backdoor, greyware). It is worth noting that, in this and the next stages, we employ (different) classifiers able to recognize misclassified benign samples, results of errors occurred in the previous stages. After being categorized, malicious files are further classified in families (e.g., *xtrat* and *vtflooder*).

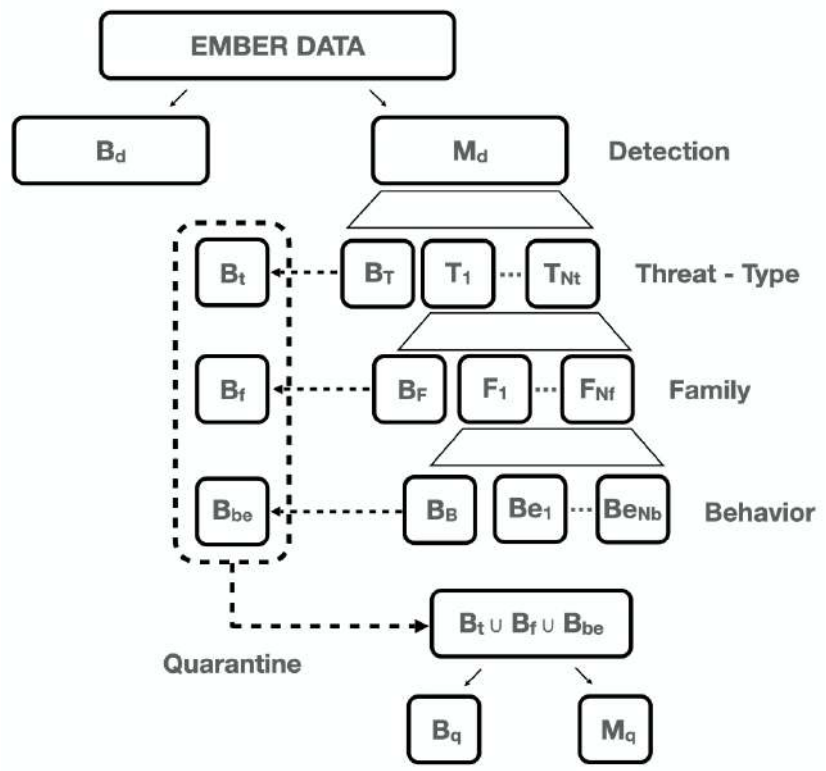


Figure 2: Representation of the proposed classification pipeline.

Analogously to the previous stage, the employed classifier is able to discriminate benign samples — identified as malicious — but also classify malicious samples belonging to less recurrent families into a specific class, called *other*. Finally, different malware families are classified according their malicious behaviour. Whenever a sample is classified incoherently through the pipeline, that is, is classified as malware is the detection stage but as a benign in some next stage, we envision another stage that gathers all these suspicious samples, the *quarantine*, and reclassifies again them to improve the overall malware detection performance.

For training pipeline's classifiers, we use the same data flow described above: samples used to validate a classifier in a stage are, then, provided in input to the next stage to train the following classifier. During the training phase, at each stage of the pipeline -detection and threat-type, family, and behavior classification-, the output of the preceding stage is split into train and validation sets, and a Gradient Boosting Decison Tree (GBDT) classifier is trained and its hyper-parameters optimized for the specific task. We used the Python API implementation of Microsoft's LightGBM[2] open source framework, that proved to be the the optimal solution between training time and performance [38].

As described in the original EMBER benchmark model [24], the raw features of each sample are mapped into a fixed-size vector of length 2351. In this work we start from the same feature set but we one slight modification. The original feature vectors rely heavily on the hashing trick in order to contain the variable and potentially very large number (order of millions) of

[2]https://github.com/microsoft/LightGBM

imported functions, while here we chose to keep only the first 151 most common ports in order to keep the balance with the other major groups of features. A comparison between our model and the original EMBER results shows that the model performance is comparable (86.3% TPR @ 0.1% FPR versus 86.8% TPR @ 0.1% FPR of EMBER's model), but with the advantage of a more interpretable result in our case.

5 Experimental Evaluation

Table 1 summarizes the experimental evaluations carried out on the proposed pipeline. For completeness, it reports both the results obtained during the validation and testing of the classifiers. Obtained results in the first stage (i.e., detection stage) are comparable with those obtained by EMBER developers [34]. It is important to note that our pipeline is able to correct false positives (benign samples detected as malicious), as discussed later in this section.

Samples predicted as malicious are then propagated to the stage that classifies threat types. Results reported in Figure 3 show that this stage has encountered the most troubles in classifying samples: indeed, benign samples are easily confused with grayware, viruses, and downloaders and the separation among different classes is not so clear (for example, as rootkits and grayware). Another aspect to take into account is the fallacy of the ground truth we used to train the threat-type stage classifier: further discussions are outlined in Section 6. Conversely to the previous stage, family classifier is more accurate in classifying specific families. More than 63% of families have been correctly classified with an accuracy greater than 85%, with many families having almost all their test samples identified by our classifier (e.g., *xtrat*, *vtflooder*, *sivis*, and *upatre*). 84% of misclassified benign samples fall in the 'other' class and this can be explained by having too few benign samples to build a classifier that is able to identify them. Further discussions are left in Section 6.

The last stage of the pipeline consists in classifying malicious samples by their behavior. For space constraints, we do not show the confusion matrix associated to the behavior stage. Similar to the threat-type classification stage, the behavior classifier fails to correctly recognize some specific behaviors (such as, *autorun* and *osmodify*). In addition to potential similar behaviors, another cause of misclassification is the few number of samples employed to train the behavior classifier: less than $20,000$ instances to train the classifier on more than 20 different behaviors.

As described in Section 4, samples classified as benign in stages after the first one are quarantined for further analyses. Of all the 200,000 total samples in the test set, only 1150 (0.57%) end up in the quarantine stage. The classifier is able to recover 221 benign samples that were misclassified in the initial detection stage and, more importantly, recovered 635 malicious samples that were incorrectly labeled as benign in one of the classification stages.

Table 1: Results of the experimental evaluation carried out on the EMBER dataset reporting accuracy, AUC, false positives, and false negatives metrics both for validation and test phases.

	Validation				Test			
Metrics	Detection	Type	Family	Behavior	Detection	Type	Family	Behavior
Samples	300,000	72,257	35,934	17,842	200,000	99,314	98,611	98,452
Accuracy	0.981	0.893	0.891	0.841	0.969	0.847	0.890	0.837
AUC	0.997	0.981	0.988	0.972	0.995	0.961	0.984	0.952
False positives	2,564	1,030	482	214	3,136	2,945	2,853	2,740
False negatives	3,225	202	78	70	3,033	512	67	181

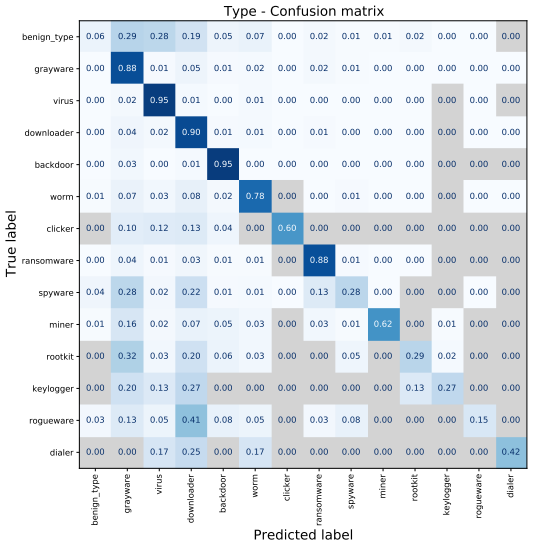


Figure 3: Confusion matrix for the threat-type classification stage reporting the performance, in terms of accuracy, on the test set described in Table 1.

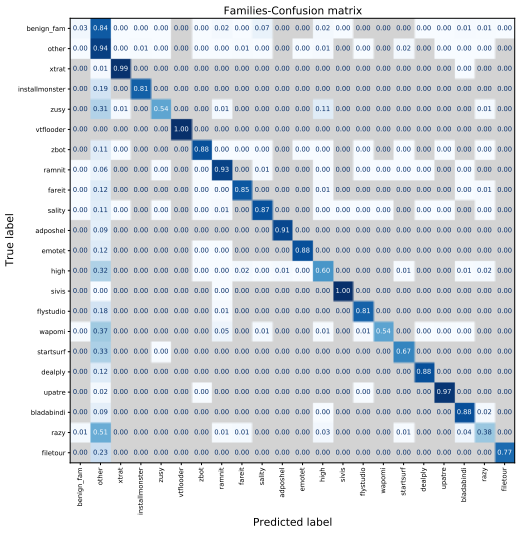


Figure 4: Confusion matrix for the family classification stage reporting the performance, in terms of accuracy, on the test set described in Table 1.

5.1 Feature Importance

The LightGBM framework [39], roughly speaking, builds a strong learner upon an ensemble of decision trees as weak learners. Decision trees assign at each learning step a dichotomous split on feature values based on the maximum obtainable information gain, so that it is a highly interpretable ML model.

Looking at table 2 we can see that features having the highest importance are common in almost all the stages, but each stage is characterized by a different ranking in the features' importance, confirming the differences in the various typology of classes and the necessity to use different models in each stage of the pipeline. Among the most frequent features we note the virtual size, that is known to be a highly discriminative feature in malware classification [40].

It is interesting to note that the most important features for the quarantine stage, that is, the last stage in which we try to recover some wrong classifications of the previous stages, belong to the major groups of the byte entropy and printable strings. Since it has been observed that entropy and the presence or not of readable strings are correlated to the presence of packed or encrypted code, this could be suggestive of the fact that the "hardest" samples to correctly classify might be packed or encrypted, a known evasion techniques and a clear limitation to an approach based solely on static analysis.

Detection	Class (Type Threat)
Section:Entropy Hashed	Header:dll charateristics Hashed
Data Directories:RESOURCE TABLE:size	Section:Entropy Hashed
General Info:Vsize	Header:timestamp
Section:Vsize Hashed	General Info:Vsize
Data Directories:RESOURCE TABLE:size	Data Directories:RESOURCE TABLE:size

Family	Behavior
Header:dll charateristics Hashed	General Info:Vsize
Data Directories:RESOURCE TABLE:size	Header:dll charateristics Hashed
General Info:Vsize	Header:timestamp
Section:Entropy Hashed	Data Directories:RESOURCE TABLE:virtual address
Strings:printabledist	General Info: n° imports

Quarantine
Byte Histogram
Strings:printabledist
Byte Entropy Histogram

Table 2: Top feature groups for the detection, threat-type, family, behaviour, and quarantine classification stages.

5.2 Model Interpretability

In order to better understand the distribution of our data in the high-dimensional original feature space and, hence, be able to identify regions in which a certain classification stage might be less reliable, we apply UMAP [41, 42], a non-linear and unsupervised dimensionality reduction technique, to the full feature vector of size 1252 keeping the first 3 components. In turn, this allows also to suggest a potential data mislabelling that could need an analyst further

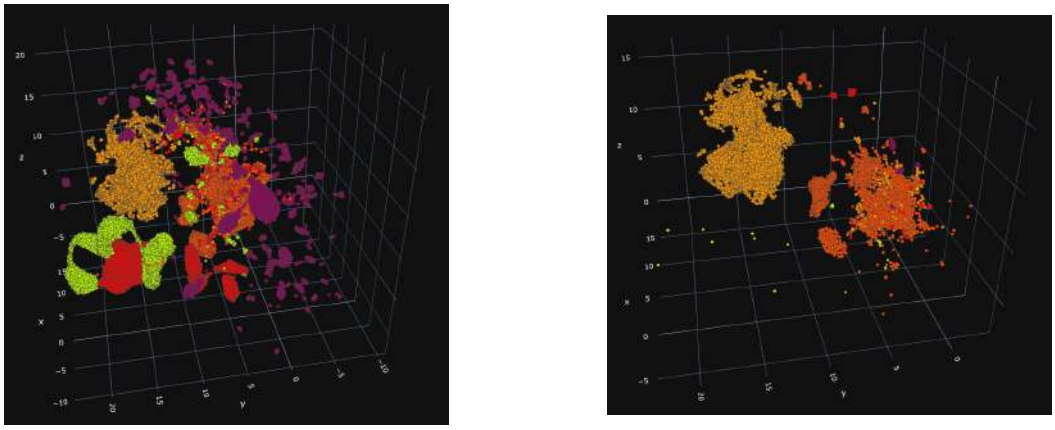


Figure 5: 3-d representation of 5 different malware families. On the left the train set (fitting phase), on the right the test set. The frequency of the classes in the train e test set are highly uneven.

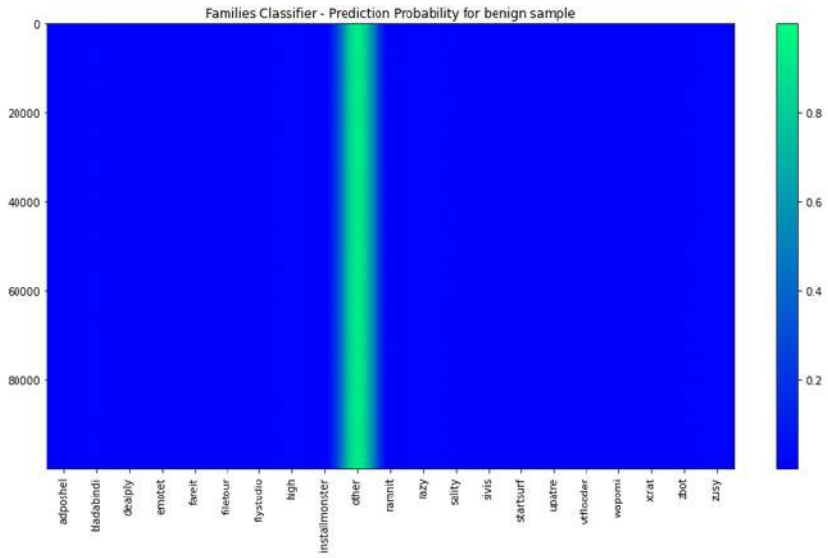


Figure 6: Prediction probability obtained by providing in input previously unseen benign samples to a family classifier, trained only on malware samples.

inspection.

In figure 5, we show the resulting embedding for the first five most frequent families. We can see that all different families are generally well separated, often forming isolated sub-clusters that can be linked to specific types or variants of the main malware class. Other regions show instead a mixing of various classes. That is where the classification is more difficult and the model is more likely to fail. There are various reasons for the appearance of this regions: first of all, as explained in section 3.2, the labels are not uniquely defined and thus it could simply be an effect of an imperfect labelling. Moreover, as pointed out in Section 5.1, many classification errors are likely to be linked to the presence of packing or encryption, that is a well known limitation to static analysis. Nevertheless, an analyst using our model could easily assess the confidence in the classification based on the position of the considered sample in this low-dimensional space, providing an useful tool for interpreting the results.

6 Discussion

As discussed in 5, some issues arise in the different stages of the pipeline due to: (*i*) mislabeling of malware families reported in the EMBER dataset (refer to Section 3), (*ii*) overlaps among different threat types and behavior, and (*iii*) few samples used to train classifiers in the last stages of the pipeline.

Regarding the second issue, we have observed that malware behaviors are not completely independent one from another: as an example, *osmodify* is often confused with *filemodify* and *execdownload* behaviors, because in AVClass it is associated with rootkit malware that, in general, establishes communications with command-and-control servers to receive new commands and malicious payloads. Finally, Table 1 reports how the number of input samples, useful to train stage classifiers, is halved at each stage. As mentioned in Section 5, few samples explain poorer performance in accuracy for family and behavior classifications.

A final consideration regards, instead, the choice of using classifiers able to discriminate between malware and benign samples to refine the overall classification process. The idea was born out of trained classifiers' capacity of accurately distinguishing between malicious and benign samples. As an example, Figure 6 shows the confidence that a different family classifier, trained only on malware samples, has in assigning previously unseen benign samples to *other* class. As already discussed in Section 5, it includes more than $2,800$ malware families that have too few samples to build a classifier able to identify them, as already mentioned in Section 3.3. It is worth noting that no benign samples has been classified as belonging to one of the most-frequent malware families of the EMBER dataset, listed on the x-axis of Figure 6. The plot has been computed on $100,000$ benign samples, extracted from the test set used for the family classification stage.

This supports our hypothesis that the poor performance in correctly detecting benign samples in the classification stages of the pipeline is due to the very small number of benign samples during model training ($\approx 1\%$ of all the benign samples).

7 Conclusion and Future Work

In this work we proposed a first implementation of a pipeline for a complete classification of Windows PE files using a machine learning approach on static features. The pipeline is able to separate benign and malicious samples, and for those samples classified as malicious it provides a exhaustive classification in terms of threat-type, malware family, and behaviour. Classification results, although suffering from known limitations such as the size of the training data, the imperfect labelling of the ground truth, and the semantic gap of models based on static features only, are comparable to the current state of the art for similar works while providing much more detailed information on malware characteristics. Finally, the extracted feature vector characterising the raw PE and the specific ML model implemented provide an interpretable result, and the pipeline is scalable to much larger datasets. Therefore, we consider this work as a first step towards a useful tool that can help security analysts to manage novel threats, reducing time and costs of the analysis. For the near future, we plan to improve the described pipeline by: (*i*) considering a larger dataset for training (*ii*) fixing the ground truth and considering the possibility of a multi-label classification scheme, and (*iii*) further explore the properties of the embedded features' space described briefly in section 5.2. It could be interesting to include also a detector for packed/encrypted samples in the early stages of the pipeline, and move to a hybrid approach, combining static and dynamic features for a better characterization of the sample files.

References

[1] Michael Sikorski and Andrew Honig. *Practical malware analysis: the hands-on guide to dissecting malicious software*. No Starch Press, 2012.

[2] Nikola Milošević. History of malware. *arXiv preprint arXiv:1302.5392*, 2013.

[3] AV-TEST malware statistics. https://www.av-test.org/en/statistics/malware/. Accessed: 2020-03-11.

[4] Rami Sihwail, Khairuddin Omar, and Khairul Akram Zainol Ariffin. A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis. *International Journal on Advanced Science, Engineering and Information Technology*, 8(4-2):1662, 2018.

[5] Ahmad Ridha Jawad, Khaironi Yatim Sharif, and Ammar Khalel Abdulsada. N/a and signature analysis for malwares detection and removal. *Indian Journal of Science and Technology*, 12:25, 2019.

[6] KA Monnappa. *Learning Malware Analysis: Explore the concepts, tools, and techniques to analyze and investigate Windows malware*. Packt Publishing Ltd, 2018.

[7] Alireza Souri and Rahil Hosseini. A state-of-the-art survey of malware detection approaches using data mining techniques. *Human-centric Computing and Information Sciences*, 8(1):1–22, 2018.

[8] Muhammad Furqan Rafique, Muhammad Ali, Aqsa Saeed Qureshi, Asifullah Khan, and Anwar Majid Mirza. Malware classification using deep learning based feature extraction and wrapper based feature selection technique. *arXiv preprint arXiv:1910.10958*, 2019.

[9] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles Nicholas. Malware detection by eating a whole exe. *stat*, 1050:25, 2017.

[10] Mansour Ahmadi, Dmitry Ulyanov, Stanislav Semenov, Mikhail Trofimov, and Giorgio Giacinto. Novel feature extraction, selection and fusion for effective malware family classification. In *Proceedings of the sixth ACM conference on data and application security and privacy*, pages 183–194, 2016.

[11] Lakshmanan Nataraj, Sreejith Karthikeyan, Gregoire Jacob, and Bangalore S Manjunath. Malware images: visualization and automatic classification. In *Proceedings of the 8th international symposium on visualization for cyber security*, pages 1–7, 2011.

[12] Lahouari Ghouti. Malware classification using compact image features and multiclass support vector machines. iet information security (01 2020), 2019.

[13] Federico Maggi, Andrea Bellini, Guido Salvaneschi, and Stefano Zanero. Finding non-trivial malware naming inconsistencies. In *International Conference on Information Systems Security*, pages 144–159. Springer, 2011.

[14] Peter Szor. *The Art of Computer Virus Research and Defense*. Pearson Education, 2005.

[15] Felipe N Ducau, Ethan M Rudd, Tad M Heppner, Alex Long, and Konstantin Berlin. Automatic malware description via attribute tagging and similarity embedding. *arXiv preprint arXiv:1905.06262*, 2019.

[16] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. Avclass: A tool for massive malware labeling. In *International symposium on research in attacks, intrusions, and defenses*, pages 230–253. Springer, 2016.

[17] Daniel Gibert, Carles Mateu, and Jordi Planes. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications*, 153:102526, 2020.

[18] Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. Survey of machine learning techniques for malware analysis. *Computers & Security*, 81:123–147, 2019.

[19] Rahul, Priyansh Kedia, Subrat Sarangi, and Monika. Analysis of machine learning models for malware detection. *Journal of Discrete Mathematical Sciences and Cryptography*, 23(2):395–407, 2020.

[20] Jagsir Singh and Jaswinder Singh. A survey on machine learning-based malware detection in executable files. *Journal of Systems Architecture*, page 101861, 2020.

[21] Edward Raff and Charles Nicholas. A survey of machine learning methods and challenges for windows malware classification. *arXiv preprint arXiv:2006.09271*, 2020.

[22] Michael R Smith, Nicholas T Johnson, Joe B Ingram, Armida J Carbajal, Ramyaa Ramyaa, Evelyn Domschot, Christopher C Lamb, Stephen J Verzi, and W Philip Kegelmeyer. Mind the gap: On bridging the semantic gap between machine learning and information security. *arXiv preprint arXiv:2005.01800*, 2020.

[23] Zhang Fuyong and Zhao Tiezhu. Malware detection and classification based on n-grams attribute similarity. In *2017 IEEE International Conference on Computational Science and Engineering*

(CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC), volume 1, pages 793–796. IEEE, 2017.

[24] P. Roth. Ember Improvements. Conference on Applied Machine Learning for Information Security (CAMLIS19), 2019.

[25] Yoshihiro Oyama, Takumi Miyashita, and Hirotaka Kokubo. Identifying useful features for malware detection in the ember dataset. In *2019 seventh international symposium on computing and networking workshops (CANDARW)*, pages 360–366. IEEE, 2019.

[26] Huu-Danh Pham, Tuan Dinh Le, and Thanh Nguyen Vu. Static pe malware detection using gradient boosting decision trees algorithm. In *International Conference on Future Data and Security Engineering*, pages 228–236. Springer, 2018.

[27] Konrad Rieck, Thorsten Holz, Carsten Willems, Patrick Düssel, and Pavel Laskov. Learning and classification of malware behavior. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 108–125. Springer, 2008.

[28] Mohamed Nassar and Haidar Safa. Throttling malware families in 2d. *arXiv preprint arXiv:1901.10590*, 2019.

[29] Xin Hu, Kang G Shin, Sandeep Bhatkar, and Kent Griffin. Mutantx-s: Scalable malware clustering based on static features. In *2013 {USENIX} Annual Technical Conference ({USENIX}{ATC} 13)*, pages 187–198, 2013.

[30] Bowen Sun, Qi Li, Yanhui Guo, Qiaokun Wen, Xiaoxi Lin, and Wenhan Liu. Malware family classification method based on static feature extraction. In *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, pages 507–513. IEEE, 2017.

[31] Bo Yu, Ying Fang, Qiang Yang, Yong Tang, and Liu Liu. A survey of malware behavior description and analysis. *Frontiers of Information Technology & Electronic Engineering*, 19(5):583–603, 2018.

[32] David Brumley, Cody Hartwig, Zhenkai Liang, James Newsome, Dawn Song, and Heng Yin. Automatically identifying trigger-based behavior in malware. In *Botnet Detection*, pages 65–88. Springer, 2008.

[33] Dmitri Bekerman, Bracha Shapira, Lior Rokach, and Ariel Bar. Unknown malware detection using network traffic classification. In *2015 IEEE Conference on Communications and Network Security (CNS)*, pages 134–142. IEEE, 2015.

[34] H. S. Anderson and P. Roth. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. *ArXiv e-prints*, April 2018.

[35] Romain Thomas. Lief - library to instrument executable formats. https://lief.quarkslab.com/, April 2017.

[36] Virus Total. https://www.virustotal.com/gui/.

[37] Silvia Sebastián and Juan Caballero. Avclass2: Massive malware tag extraction from av labels. In *Annual Computer Security Applications Conference*, pages 42–53, 2020.

[38] C. Galen and R. Steele. Evaluating performance maintenance and deterioration over time of machine learning-based malware detection models on the ember pe dataset. In *2020 Seventh International Conference on Social Networks Analysis, Management and Security (SNAMS)*, pages 1–7, 2020.

[39] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30:3146–3154, 2017.

[40] J. Yonts. White paper: Attributes of malicious files. Technical report, SANS Institute, June 2012.

[41] L. McInnes, J. Healy, and J. Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *ArXiv e-prints*, February 2018.

[42] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Grossberger. Umap: Uniform manifold approximation and projection. *The Journal of Open Source Software*, 3(29):861, 2018.

ITASEC

ITALIAN CONFERENCE ON CYBERSECURITY

Ancona, Italy
February 4th – 7th, 2020

What is ITASEC?

The **Italian Conference on Cyber Security** is the most relevant conference dedicated to cyber security at the national level and it is organized annually by **CINI**, the **Italian National Cyber Security Lab**. The 2020 edition will dedicate the first day to **scientific workshops** and **tutorials**, while the conference program will be rich with ad hoc sessions dedicated to **academic paper** presentations, **vendor spaces** and **vision speeches** provided by sponsor companies. The main cyber security related themes include Blockchain, Cryptology, Data Security and Privacy, Security Management and Governance, Operational Incident Handling and Digital Forensics, AI and Security.

Our contribution

aizoOn will present a scientific paper titled "*DNS Covert Channel Detection via Behavioral Analysis: a Machine Learning Approach*", written by our colleagues from the Aramis teams and already discussed at the MALCON conference, hosted in October 2019 in the USA.

Detecting covert channels among legitimate traffic represents a severe challenge due to the high heterogeneity of networks. Therefore, we propose an effective covert channel detection method, based on the analysis of DNS network data passively extracted from the network monitoring system aramis.

This technique is based on a machine learning module and on the extraction of specific anomaly indicators able to describe the problem at hand. The contribution of this paper is two-fold: (i) the machine learning models encompass network profiles tailored to the network users, and not to the single query events, hence allowing for the creation of behavioral profiles and spotting possible deviations from the normal baseline; (ii) models are created in an unsupervised mode, thus allowing for the identification of zero-days attacks and avoiding the requirement of signatures or heuristics for new variants.

The proposed solution has been evaluated over a real network in 15-day long experimental session, with the injection of 7 different APTs and 1 PoS malware campaign and the network traffic of 5 DNS-tunneling tools and 1 DNS-file-transfer tool; all the malicious variants were detected, while producing a low false-positive rate during the same period.

MALCON IEEE

INTERNATIONAL CONFERENCE ON MALICIOUS AND UNWANTED SOFTWARE (MALWARE)

Nantucket, Massachussets – USA

October 1st -4th, 2019

What is MALCON?

MALCON is a conference that brings together industry practitioners and researchers, together with academicians and law enforcement officials with the aim of **understanding, anticipating and creating defense systems against emerging threats and novel attacks**. This year, MALCON keynote speeches and presentations will tackle the analysis of the main **challenges of the computer security industry**, as well as the considerations concerning **malware evolution** in the era of mobile devices and Cloud and, ultimately, the application of **reverse engineering** and similar techniques.

Our contribution

aizoOn presented a **scientific paper** titled "*DNS Covert Channel Detection via Behavioral Analysis: a Machine Learning Approach*", written by our colleagues from the Aramis teams.

Detecting covert channels among legitimate traffic represents a severe challenge due to the high heterogeneity of networks. Therefore, we propose an effective covert channel detection method, based on the analysis of DNS network data passively extracted from the network monitoring system **aramis**.

This technique is based on a machine learning module and on the extraction of specific anomaly indicators able to describe the problem at hand. The contribution of this paper is two-fold: (i) the machine learning models encompass network profiles tailored to the network users, and not to the single query events, hence allowing for the creation of behavioral profiles and spotting possible deviations from the normal baseline; (ii) models are created in an unsupervised mode, thus allowing for the identification of zero-days attacks and avoiding the requirement of signatures or heuristics for new variants.

The proposed solution has been evaluated over a real network in 15-day long experimental session, with the injection of 7 different APTs and 1 PoS malware campaign and the network traffic of 5 DNS-tunneling tools and 1 DNS-file-transfer tool; all the malicious variants were detected, while producing a low false-positive rate during the same period.

DNS Covert Channel Detection via Behavioral Analysis: a Machine Learning Approach

Salvatore Saeli, Federica Bisio, Pierangelo Lombardo, Danilo Massa
aizoOn Technology Consulting
Strada del Lionetto 6, 10146 Turin, Italy
{salvatore.saeli,federica.bisio,pierangelo.lombardo,danilo.massa}@aizoongroup.com

Abstract

*Detecting covert channels among legitimate traffic represents a severe challenge due to the high heterogeneity of networks. Therefore, we propose an effective covert channel detection method, based on the analysis of DNS network data passively extracted from a network monitoring system. The framework is based on a machine learning module and on the extraction of specific anomaly indicators able to describe the problem at hand. The contribution of this paper is two-fold: (*i*) the machine learning models encompass network profiles tailored to the network users, and not to the single query events, hence allowing for the creation of behavioral profiles and spotting possible deviations from the normal baseline; (*ii*) models are created in an unsupervised mode, thus allowing for the identification of zero-days attacks and avoiding the requirement of signatures or heuristics for new variants. The proposed solution has been evaluated over a 15-day-long experimental session with the injection of traffic that covers the most relevant exfiltration and tunneling attacks: all the malicious variants were detected, while producing a low false-positive rate during the same period.*

1 Introduction

One of the most serious threats to the current society is represented by cybercrime, with heavy and sometimes dramatic consequences for many companies, organizations and single individuals [27, 28, 38, 55, 60]. Data exfiltration, in particular, plays a key role in the cybercrime scenario, as it is related to the stealing of sensitive information [26]. Among different techniques of exfiltration, *covert channels* represent a significant threat for defenders, as they are widely used and their detection is challenging. A covert channel [62] can be defined as a way to communicate, transfer or exfiltrate data while exploiting network resources never intended for this purpose. The aim of such a technique is to extract sensitive information from organizations and companies, while eluding conventional security measures (e.g., intrusion detection systems and firewalls).

Among the reasons that make covert channels a severe menace for threat hunters, it is worth mentioning the following: (*i*) conventional intrusion detection and firewall systems typically fail in the detection of covert channels; (*ii*) the network traffic varies considerably, thus causing detection issues for classical statistical approaches to covert channel detection; (*iii*) related to the previous two points, another issue is the difficulty in distinguishing covert channels between legitimate communications, this is often caused by the absence of focus on users behavioral analysis; (*iv*) although many works focus on the process of tunnel attacks, only a very restricted number of them analyzes the properties useful to describe the data exfiltration process.

In this paper, we propose an effective technique for the detection of DNS covert channels, based on the analysis of network data passively extracted by a network monitoring system. The proposed framework is based on a machine learning module and on the extraction of specific anomaly indicators able to describe the problem at hand. The focus of the machine learning module is the creation of models embedding the behavioral characteristics of a user and hence spotting possible deviations from the normal baseline. The power of this approach is the ability to identify zero-days attacks, without the requirement of signatures or heuristics for new variants. However, it may carry the drawback of highlighting also non-malicious events that are similar to covert channels from the perspective of DNS behavior. This could lead to a large number of false positives, but it is mitigated by a subsequent *advanced analytics* module, which encodes cyber security knowledge. This structure allows the proposed framework to identify a dual typology of attacks: (*i*) *exfiltration*, and (*ii*) *tunneling*.

The remainder of the paper is structured as follows. Sect. 2 provides an overview of the related work. In Sect. 3 we briefly describe the monitoring platform containing the

covert channel detection method, which is the focus of this paper. After an introduction on the problem, given in Sect. 4, the proposed approach is described thoroughly in Sect. 5. Finally, we discuss the experimental results in details in Sect. 6 and possible future developments in Sect. 7.

2 Related Work

State-of-the-art works regarding covert channels usually distinguish between two different types: (*i*) storage covert channels [13], where covert bits are strictly bounded to the communication protocols under analysis (e.g., IP, DNS, HTTP, SMB, SSL); (*ii*) timing covert channels [52], based on the manipulation of timing or on the ordering of network events (e.g., packet arrivals).

Depending on the type of covert channel, associated detection technique may also vary: (*i*) for the storage covert channels, typical detection methods involve Markov Chains and Descriptive Analytics [12]; (*ii*) for the timing covert channels, many different approaches have been considered, in particular: statistical tests of traffic distribution [47], regularity tests of time variations within the traffic [22], and machine learning methods [50] such as Support Vector Machines (SVMs) [3] and Bayesian Networks [2].

The detection framework proposed in this article focuses on storage covert channel, while borrowing some detection techniques typically used for the detection of timing covert channels, e.g., the use of SVMs. For the proposed technique, DNS covert channels are taken into consideration. This is motivated by the fact that, at the time of this writing, DNS represents one of the most common protocol to control systems and exfiltrate data covertly [11, 17, 43]. Nevertheless, DNS covert channel is a method that many organizations still fail to detect. In its simplest form, this technique employs the DNS protocol to communicate directly with an attacker's external DNS server.

DNS is not designed to exchange an arbitrary amount of data: therefore, messages are usually short and answers are not correlated and may not be received in the same order as the corresponding requests. Usually attackers avoid these limitations with two approaches: (*i*) *tunneling* [39]: the attacker establishes a bidirectional channel to send communications and instructions from an external server (command and control, or C&C) to a compromised host; (*ii*) *exfiltration* [44]: the attacker executes data exfiltration from a compromised host towards a controlled external server, sending information with minimal overhead and short and independent requests.

Ahmed et al. [1] analyzed data extracted by a network monitoring system, as several other works in state-of-the-art [3, 35, 41]. The authors used only stateless attributes of individual DNS queries, based on three main categories as characters count (e.g., total count of characters in FQDN, count of characters in subdomains, count of uppercase and numerical characters), entropy on strings, and length of discrete labels in the query (e.g., maximum label length and average label length). The authors developed an anomaly detection system based on Isolating Forest (iForest). The approach does not involve any particular DNS record type, the test phase is deployed solely with the exfiltration tool DET and the training phase of the model is computationally very expensive.

Nadler et al. [44] proposed an approach based on the characteristics of the queries employed for data exchange over DNS (e.g., longer than the average requests and responses, with encoded payload and a plethora of unique requests) and on the use of single domains for exfiltration. As in the previous case, the anomaly detection was based on iForest. However, this approach considers only A and AAAA records usage, the tested malware was simulated with ad-hoc crafted queries in experimental environment; iodine and dns2tcp were the only tunneling tools taken into account.

Das et al. [15] underlined that DNS tunneling and malware exfiltration typically involve an exchange with the attacker server of a part of payload encoded in a subdomain portion of the DNS query or in the response packet. The authors proposed two machine learning approaches, (*i*) a logistic regression model and (*ii*) a k-means clustering, respectively (*i*) for the exfiltration and (*ii*) for the tunneling scenarios; these approaches are based on grammatical features extracted from queries representative of an encoded payload (e.g., entropy and number of upper cases, lower cases, digits, and dashes characters). However, the test phase lacks several attack scenarios; in particular, for DNS tunneling, only the tunneling tool dnscat2 with the TXT record was employed.

Liu et al. [39] analyzed the traffic of several open-source DNS-tunneling tools based on the extraction of four kinds of features, including time-interval features (e.g., mean and variance of time-intervals between a request and a response), request packet size, domain entropy, and distinction of record types (e.g., A, TXT, MX). The authors developed a binary supervised classification model based on the description of traffic generated from different DNS-tunneling tools (e.g., dnscat2, iodine, and dns2tcp). The solution proposed by [39] consists of an offline component, where the system trains the classifier, and an online component, where the system identifies the tunnel traffic. Nevertheless, during the training phase the DNS traffic is not collected in a streaming fashion but using a specific dataset.

3 Network Monitoring Platform

The proposed covert channel detection algorithm has been deployed in a network security monitoring platform

called *aramis* (Aizoon Research for Advanced Malware Identification System) able to automatically identify a wide range of malware and attacks in near-real-time. This software is bundled with dedicated hardware[1], and its structure can be summarized in four phases:

1. Collection: sensors are placed in various nodes of the network. Each sensor gathers the data from its segment of the network, pre-analyzes them in real-time and sends the results to a NoSQL database.
2. Enrichment: inside the NoSQL database, data is enriched with information coming from the Cloud Service, which collects intelligence from various OSINT sources and from internally managed sources.
3. Analysis: two kinds of analyses are executed on the stored data: (i) *advanced cybersec analytics* to spot and highlight specific attacks, among which the covert channel detection module can be found, and (ii) a *machine learning engine* which compares the behavior of each node with the usual one.
4. Visualization: the results are presented through cognitive dashboards, crucial to highlight anomalies.

4 A Glimpse into DNS Covert Channels

In DNS covert channel technique the main challenge is represented by performing the data exchange in an optimized manner. To accomplish this goal, the structural and grammatical characteristics of hostname and the capabilities of various DNS record types are exploited [24]. According to the RFC 1034, the hostname can be up to 253 characters long and it is composed by labels each of which can be up to 63 characters long; each character can be a letter (upper case and lower case are both permitted), a number or a hyphen. To maximize the data exchanged - the payload transmitted - in each DNS query, each label of the subdomain contains a portion of the data, which have previously been encoded (e.g., using Base32, Base64, Base128 or Hexadecimal codecs) or encrypted (e.g., using RC4 encryption algorithm). Furthermore, depending on the DNS covert channel scenario, specific DNS record types are employed. We distinguish between two different cases: (*i*) *exfiltration*: information might be contained in a subdomain of a domain and usually the communication is based on A or AAAA DNS record types; (*ii*) *tunneling*: a bidirectional channel between a compromised host and a server controlled by the attacker is established in order to send commands and obtain information about the host. The requirement to maintain an open connection determines the choice of certain DNS records – such as TXT (which is the most common choice), KEY, CNAME, MX, SRV, NULL, and PRIVATE – that allow transmitting arbitrary portions of text, in addition to the information exchanged over the subdomain.

Another crucial point is represented by the information stored in the stub revolver cache. In fact, normal DNS traffic is usually reduced by the fact that a huge amount of DNS responses are cached within the stub resolver; instead, when a covert channel takes place, the domain-specific traffic tends to avoid cache by using non-repeating or short time-to-live messages, resulting in not repeated and unique queries [24].

5 Detection of DNS Covert Channels

The proposed DNS covert channel detection technique is based on the monitoring of large volumes of DNS requests from a given IP address and on the analysis of the domain-related linguistic features [7]. The main idea is to employ machine learning to create models able to embed the behavioral characteristics of network users' traffic. An event is considered anomalous when its results are distant from the typical behavior modeled by the algorithm. Note that the analysis relates different queries, and therefore the models encompass network profiles tailored to the network users, and not to the single query events. Furthermore, the possible risk of false positives generation related to the use of machine learning is mitigated by the analysis of specific descriptors of the DNS tunneling or exfiltration process built out of the network data.

The general process of the resulting framework is shown in Figure 1. The raw local recursive DNS server (RDNS) data are collected, parsed and transformed into logs by the network monitoring platform described in Sect. 3. Logs are filtered as described in Sect. 5.1, both in the offline and in the online component: the first one periodically[2] extracts the features described in Sect. 5.2 from the historical data collected during the period between two subsequent runs and builds the related machine learning models. These models are then used by the online component, which extracts the related features, detects anomalous queries in real time, and further analyzes them in order to spot suspicious queries, which will be the final outputs of the algorithm, as described in Sect. 5.3. In the following, the three main phases that compose the algorithm are summarized: *(i) filters*: filtering of input RDNS queries extracted by a network analyzer, *(ii) offline phase*: assessment of the network under analysis and creation of models able to describe the normal behavior of the network, *(iii) online phase*: validation of such models to spot potentially anomalous and/or malicious activities occurring in the network.

[1] E5-2690 2.9GHz x 2 (2 sockets x 16 cores) 16 x 8GB RAM, 1.1TB HDD

[2] The period was set to six hours as a tradeoff between the need to analyze enough data to build reliable models and the need to avoid the abuse of hardware resources.

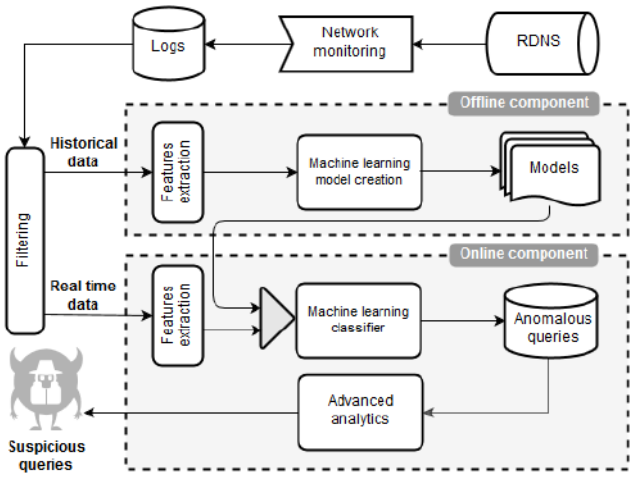


Figure 1. Diagram representing the main steps of the detection algorithm: (*i*) filters, (*ii*) offline phase, and (*iii*) online phase.

Table 1. Filters Description

Type	Description
White list domains	Domains known to be trusted, e.g., the top 10000 domains in the world provided by Alexa [4] and the web URLs of the 500 world biggest companies provided by Forbes [25].
Response code	Only communications which occurred with no error (associated with response code=0) are kept and the analysis of retransmissions due to network errors is avoided.
Content Delivery Networks	These represent an important source of false positives for DNS-based detection algorithms [40].
Overloaded DNS	DNS queries are often overloaded so to provide anti-spam or anti-malware techniques.
Local and corporate domains	These domains represent a high percentage of the legitimate DNS traffic in a corporate network.
IP addresses in the subdomain	Domains containing an IP address in the subdomains are typically associated with answers to PTR requests.
Longest label with less than 6 characters	We assume that the longest labels used to transfer a payload necessitate of a minimum number of characters.
Less than 3 different hostnames per domain	After grouping queries with the same domain, we select domains associated with at least 3 distinct hostnames, in order to discard the ones which cannot be involved in an exfiltration process (due to the minimum number of queries involved in the process).
Duplicated queries	Duplicated queries are mainly related with retransmissions, quite common in complex networks.

5.1 Filters

The input for both phases is represented by DNS queries passively extracted by the network analyzer described in Sect. 3. In order to consider the traffic that may actually be associated with exfiltration or tunneling, the following DNS query types are analyzed: TXT, CNAME, MX, SRV, NULL, KEY, A, AAAA. The collected data are filtered as described in Table 1: any query or group of queries matching any of the filters is removed from the subsequent analysis.

5.2 Offline Phase

The offline phase takes as input the historical data collected from the network and representing a baseline of the normal behavior. This phase includes:

1. *Feature extraction*: raw features are parsed in order to extract the valuable information useful to describe the problem at hand.
2. *Model creation*: a classifier is trained using the extracted features, with the aim of distinguishing between patterns performing with a normal behavior and potentially suspicious patterns.

5.2.1 Feature Extraction

The features used for training the classifier are the following.

Uppercase Characters Ratio. Ratio between the number of uppercase characters and the total number of characters in the subdomain. A large fraction of uppercase characters may be associated with a payload encoded in the subdomain using Base64 encoding, which is widely employed in DNS covert channels.

Digits Ratio. Ratio between the number of digits and the total number of characters in the subdomain. A large fraction of digits may indicate a payload encoded or encrypted into the subdomain.

Total Label Ratio. Total number of characters of the hostname divided by 253 (maximum number of characters allowed for a hostname according to RFC 1034). In a wide variety of DNS covert channel scenarios, a payload is encoded or encrypted in the subdomain, aiming at transferring the maximum possible amount of data to conduct the attack. As a result, the obtained queries possess a larger number of characters compared with legitimate queries.

Per Label Ratio. Number of characters of the longest label of the subdomain divided by 63 (maximum number of characters allowed for each label in the hostname according to RFC 1034). The encoded or encrypted payload might contain several types of information (e.g., bot-id, campaign-id, command), in addition to the data exchanged. As a result, one or more labels, used to exchange data, may contain a large fraction of the characters in the subdomain, while in ordinary traffic the lengths of the labels are typically comparable, resulting in a moderate length for the longest label.

5.2.2 Model Creation

A one-class Support Vector Machine (SVM) [51] classifier with radial basis function kernel has been used to create the models out of the feature space described in the previous subsection. In fact, even though the original formulation of SVMs is related to the resolution of supervised tasks, the one-class SVM – which has been shown to be an appropriate choice in the context of anomaly detection [54] – is defined as a boundary-based anomaly detection method, which modifies the original SVM approach by extending it in order to deal with unsupervised data. In our context, this means that the proposed approach is able to train the classifier by using only the normal network traffic, preserving the malicious samples for the test of the algorithm. In particular, this implies that the proposed method is more likely to identify a new variant of a DNS covert channel, as it does not require a specific training for that variant.

Like traditional SVMs, one-class SVMs can also be extended to non-linearly transformed spaces using the so called kernel trick, which amounts to define an appropriate scalar product in the feature space. In the present work a *radial basis function* kernel has been used for the reasons described in [36], i.e., the scalar product between two features vectors $\vec{x}$ and $\vec{x'}$ has been defined as in Eq. 1

$$K(\vec{x}, \vec{x'}) = \exp(-\gamma||\vec{\mathrm{x}} - \vec{\mathrm{x}}'||^2), \quad (1)$$

where γ is a hyper-parameter which defines the width of the Gaussian distribution. The objective function maximized by the algorithm is the so-called *soft margin*, which is characterized by ν, another hyper-parameter associated with the penalty related to wrong labelling [51].

The model creation phase (see, for example, [45] for an introduction on the topic) is repeated every six hours: each time the collected historical data are randomly split in two subsets. The first subset (*training set*) contains 75% of the data and it is used to train the model over the grid $(\gamma, \nu) \in \{10^{-3}, 10^{-2}, \cdots, 10^{2}\} \times \{10^{-3}, 10^{-2}, \cdots, 10^{2}\}$. The latter subset (*validation set*) contains the remaining 25% of the data and it is used to select the best pair (γ, ν),[3] which is used to recreate the model on the whole data set (*training* + *validation*).

5.3 Online Phase

The online phase analyzes real time data and involves the following steps:

1. *Feature extraction*: the same features described in Sect. 5.2 are extracted.
2. *Classification*: the classifier trained in the offline phase is applied to online data in order to check whether they are consistent with the normal behavior; if they do not conform, the patterns are assessed with the subsequent module.
3. *Advanced analytics*: an algorithm combines data mining techniques and cyber security knowledge to perform an in-depth analysis of the queries considered suspicious in the previous step. In the remainder of the section, we provide a description of this module.

[3] Although in principle a different value for the pair (γ, ν) is allowed in every model creation phase, the validation procedure has selected $(\gamma, \nu) = (0.1, 0.1)$ for the whole duration of our experiment.

5.3.1 Advanced analytics

The following anomaly indicators of possible covert channel activity are developed.

Number of unique requests from an IP address to a domain. Avoiding the stub resolver cache during an attack may result in not-repeated requests to a domain. Therefore, a large number of unique requests from an IP address to a domain may indicate a DNS covert channel. An indicator i_r is defined as the frequency with which the number of unique requests from an IP address to a domain falls on the tails of its distribution (90th percentile).

Number of unique hostnames per domain. Related to the previous point, also a large number of unique hostnames per domain might indicate a DNS covert channel. An indicator i_h is defined as the frequency with which the number of subdomains per domain falls on the tails of its distribution (90th percentile).

Entropy. The entropy of a subdomain considered as a sequence of characters is related to its randomness. Since attackers compress the data to be sent as a payload via encoding or encryption, a large entropy may be a sign of an encoded or encrypted payload. An indicator i_e is defined as the maximum among the entropy of the whole subdomain and that of its longest label.

Distribution of frequencies of mono-grams and bi-grams. A distribution of subdomain characters, which is distant from the distribution of characters of real languages, may represent another indicator of randomness [48], related to an encoded or encrypted payload. For each considered language (English and Italian), $i_d^{\mathrm{lang,mono}}$ ($i_d^{\mathrm{lang,bi}}$) is defined as the maximum among the Jaro-Winkler distance [59] of the mono-grams (bi-grams) distribution of the subdomain and that of its longest label from the corresponding distribution of the language. Finally, we evaluate the average $i_d = (i_d^{\mathrm{eng,mono}} + i_d^{\mathrm{eng,bi}} + i_d^{\mathrm{ita,mono}} + i_d^{\mathrm{ita,bi}})/4$.

Anomaly Index. Once the previously described indicators tailored to the DNS protocol are extracted, an *anomaly index* A is built by averaging them: $A = (i_r + i_h + i_e + i_d)/4$. For each machine source, the index is rescaled by taking into account the fraction between the number n_s of suspicious queries and the total number n_{tot} of DNS queries pro-

Table 2. Malware Description

Name	Category	APT	Codec	DNS rtype	Domain	Detection	A
Pisloader	Trojan RAT	Wekby	Base32	TXT	local.it-desktop.com	96%	100%
ISMDoor	Trojan RAT	GreenBug	Base64	AAAA	basnevs.com	91%	100%
Denis	Trojan Backdoor	OceanLotus	Base64	NULL	z.teriava.com	100%	100%
Carbanak	Trojan Backdoor	Fin7	Custom	TXT	en.google4-ssl.com	100%	100%
Cobalt Strike	Commercial Tool	CopyKittens	Custom	TXT	update.cisc0.net	100%	100%
Bondupdater	Trojan Powershell	OILRig	Custom	A, TXT	withyourface.com	100%	100%
UDPoS	PoS Malware	-	RC4	A	ns.service-logmeln.network	100%	100%
DNSpionage	Trojan RAT	-	Base32	A	microsoftonedrive.org	100%	100%

duced, according to $A \to \min(1, \mathrm{A} + \mathrm{b} + \mathrm{c}\log(\mathrm{n_s}/\mathrm{n_{tot}}))$, where $(b, c) = (0.33, 0.067)$ have been set with a preliminary *una tantum* validation procedure. The detection of covert channels has thus been reduced to a very simple one-dimension classification problem: only queries with $A > A_{\mathrm{th}}$ are labeled as suspicious, where the optimal threshold ($A_{\mathrm{th}} = 0.25$) has been found with the *una tantum* validation procedure. Note that A may be considered as an indicator of anomaly from a behavioral point of view.

Table 3. Network Description

	15-Days Total	1-Hour Average
N. of Machines	360	-
N. of Client Machines	346	-
N. of Connections	43 M	287 k
N. of Resolved DNS Queries	4 M	25 k
N. of Unique Resolved DNS Queries	119 k	791

6 Experimental Evaluation

The proposed DNS covert channel detection algorithm was evaluated over a real company network: in particular, the test set comprises 15 days of ordinary traffic – described in Table 3 – with the injection of traffic which covers the most relevant exfiltration and tunneling attacks. Note that the test set has been only used to test the performance of the algorithm and not to modify the algorithm or the parameters.

Two different experimental designs have been selected for the test of the algorithm, related with both *exfiltration* and *tunneling* attacks: a malware scenario and a tool scenario, that correspond to two different modes of injection described in Sect. 6.1 and 6.2 respectively. Both scenarios have been recreated by injecting the malicious traffic into the ordinary traffic of a real network, described in Table 3.

6.1 Malware Scenario

The traffic related to malware attacks was injected by employing 8 pcaps – collected from the public sandboxes [31, 33, 49] – associated with 7 different APTs [14, 23, 29, 42, 53, 57, 58] and 1 PoS malware campaign [56]. Table 2 provides a brief description of each malware with the following information:

- The malware name;
- The category of the malware;
- The name of the APT group related to the malware;
- The codec employed to encode the payload into the DNS requests;
- The list of DNS record types used by the malware;
- The domain present in each pcap, which is a known IoC associated to the malware;
- The percentage of detected occurrences, i.e., the number of detected queries divided by the total number of queries produced by the malware;
- The value of the anomaly index A, defined in Sect. 5.

6.2 Tool Scenario

The traffic related to attacks performed by tools was injected in the network described in Table 3 by using 5 of the most popular DNS-tunneling tools [16, 18, 19, 20, 32, 61] and 1 DNS-file-transfer tool [21]. This choice is motivated by the state-of-the-art in the exploitation of DNS protocol for covert channel, assessed by the presence of these tools inside some Linux distributions for penetration testing, the supported DNS record types and the state of maintenance of the tools. Table 4 provides a brief description of each tool with the following information:

- The name of the tool;
- The list of compatible platforms;

Table 4. Tools Description

Name	Platform	Linux PenTest Distro	Codec	DNS rtype	Detection	A
dnscat2	Linux, Windows	-	Hexadecimal	TXT	100%	100%
				MX	100%	100%
				CNAME	100%	100%
dns2tcp	Linux, Windows	Kali, BlackArch	Base64	TXT	100%	100%
				KEY	100%	100%
iodine	Linux, Windows, Mac OS X	Kali, BlackArch, BackBox	Base128	NULL	70.08%	100%
				TXT	87.02%	100%
				SRV	85.03%	100%
				MX	88.11%	100%
				CNAME	77.73%	100%
				A	86.67%	100%
DNScapy	Linux, Mac OS X	-	Base64	CNAME	13.73%	69%
				TXT	12.61%	71%
				TXT, CNAME	14.75%	76%
dnsfilexfer	Linux, Windows, Mac OS X	Kali, BlackArch	Hexadecimal	A	100%	100%
Your-Freedom	Linux, Windows, Mac OS X	-	Base64	NULL	100%	100%

- The Linux penetration testing distribution where the tool is pre-installed;
- The codec employed to encode the payload into the DNS requests;
- The list of DNS record types supported by the tool;
- The percentage of detected occurrences, i.e., the number of detected queries divided by the total number of queries produced by the malware;
- The value of the anomaly index A, defined in Sect. 5.

All the tools considered in this paper require two different machines: (*i*) a host which represents a server hold by an attacker and (*ii*) another host which represents either an unaware infected client or a client controlled by an insider threat. Both components have been recreated in an appropriate way in each experiment (see Table 4).

1. The server component was simulated, whenever possible, with an appropriate version of a Linux penetration testing distribution such as Kali Linux [34], BlackArch Linux [8], and BackBox [6]. In the remaining cases an *ad hoc* Linux installation was employed;
2. The client component was simulated with a Windows or Linux client, with an appropriate version of the operative system (for reasons of compatibility with installed tools).

All tools employed in the experimental evaluation are open-source except Your-Freedom. A brief description of the tools considered follows.

iodine [32] is one of the most popular DNS-tunneling tools [9, 10, 30, 39, 44] and is pre-installed in all Linux penetration testing distributions considered in this paper. Iodine encapsulates an IPv4 packet into the payloads of DNS packets and needs a TUN/TAP device to operate. It uses the NULL record type by default, but can support other record types such as PRIVATE, TXT, SRV, MX, CNAME and A (returning CNAME). Upstream data is GZIP compressed and encoded; the supported encoding option includes Base32, Base64, Base64URL and Base128. If NULL or PRIVATE record types are used, downstream data is transmitted as GZIP compressed raw IP packet bytes; if other record types are used, it is GZIP compressed and encoded like upstream data.

dnscat2 [20, 39, 48] is designed to create an encrypted command-and-control (C&C) channel over the DNS protocol. It uses the TXT, CNAME and MX record types by default, but it supports also A and AAAA record types if data are only sent from client to server. All data in both directions is transported using hexadecimal encoding.

dns2tcp [9, 10, 18, 39] relays TCP connection over DNS, and is pre-installed in Kali Linux and in BlackArch Linux. Data encapsulation already takes place at the TCP level, so no separate driver (TUN/TAP) is required. Dns2tcp requires the list of available resources (e.g., ssh, smpt, pop3) on server configuration, i.e., the resources that the client can request to access. It uses the TXT record type by default, but it can also support the KEY record type. All data in both directions is transported using Base64 encoding.

DNScapy [10, 19] creates a SSH tunnel through DNS

packets. The idea of encapsulating SSH in DNS comes from OzymanDNS [46]. DNScapy supports CNAME and TXT record types but the default mode is RAND, which randomly employs both CNAME and TXT. All data in both directions is transported using Base64 encoding.

dnsfilexfer [21] exfiltrates files via DNS lookup and is pre-installed in Kali Linux and in BlackArch Linux. Dnsfilexfer supports only the A record type. All data is transported using hexadecimal encoding.

Your-Freedom [5, 37, 61] is a tool based on a service available either in a paid version or in a free version, with some limitations. Only the client component can be downloaded and it requires either OpenVPN or a software that acts as a *socksifier*. An appropriate online server, accessible by the client component, has to be chosen during the setup of the client. Your-Freedom supports many DNS record types such as NULL, WKS, TXT, CNAME and MX. All data in both directions is transported using Base64 encoding.

6.3 Experimental Results

The DNS covert channel detection algorithm described in Sect. 5 has been evaluated over a 15-day-long experimental session performed in a test network (described in Table 3) with the injection of traffic related to several malware and tool attacks. Tables 2 and 4 (described in Sects. 6.1 and 6.2 respectively) clearly show that the proposed method successfully detected all the covert channel attacks with high anomaly and detected occurrence indicators. In particular, all tools described in Table 4 have been detected with high detection rate with the exception of DNScapy. In this context, the main difference between DNScapy and the other variants is the fact that the former produces many state queries to maintain the established connection alive; these queries are shorter and are not identified by the algorithm, which focuses on exfiltration queries. Anyway, it is important to note that the proposed method is indeed able to detect a covert channel attack via DNScapy, as the exfiltration queries are correctly detected. The main purpose of the algorithm is therefore fully reached despite the fact that the detection rate on the queries is low for this particular tool.

In Table 5 we summarize the performance of the algorithm via the confusion matrix, which contains:

- True Negatives (T_N): The number of unique legitimate queries that are correctly labeled as legitimate.
- False Negatives (F_N): the number of unique queries related with malicious covert channels queries that are incorrectly labeled as legitimate.
- False Positives (F_P): the number of unique legitimate queries that are incorrectly labeled as covert channels; many of them are related with particular types of advertising which produce queries that look very similar to covert channels[4].
- True Positives (T_P): the number of unique queries related with malicious covert channels that are correctly detected.

Table 5. Results (Confusion Matrix)

		Actual Class	
		Normal	**Malicious**
Predicted Class	**Normal**	$T_N = 116649$	$F_N = 490$
	Malicious	$F_P = 2033$	$T_P = 18174$

A remarkable result is the low rate of false negatives: this determines indeed a 97% recall, also known as detection rate, $R = T_P/(T_P + F_N)$. In particular, false negatives are mainly due to the evasion of the state queries related to the DNScapy tool – as previously explained in this Section – while false positives are mostly related with advertising queries.

Another metric commonly used to evaluate binary classifiers is the F-score. It is defined as $F = 2\,P\,R/(P + R)$ (where $P = T_P/(T_P + F_P)$), and it is a more reliable indicator for problems in which the classes are highly unbalanced, as in the present case (the number of legitimate queries in the test set is much larger than the number of covert channel related queries). The value obtained in the experiments is $F = 94\%$. Even though an exact comparison is not possible due to the use of different datasets, we can note that we obtained an F-score almost as large as the one in the work of [39], despite some crucial differences: (*i*) the method proposed in [39] employs a supervised classifier, i.e., a binary classifier is trained using both legitimate and malicious queries, while our approach trains the classifier only with the normal network traffic; this means that our approach can identity a new variant of a DNS covert channel without a specific training for that variant; (*ii*) the benign parts of the training and test sets in [39] only contain domains in the Alexa top 1-million list [4], while our corresponding sets contain all the traffic collected from a real network. We can therefore conclude that the proposed method brings a relevant contribution in the state-of-the-art of DNS covert channel detection.

7 Conclusions

In this paper, we proposed a DNS covert channel detection method based on the analysis of the DNS traffic of a

[4]Here are some examples of false positive: (*i*) "*0w57c49k-db0dd2cc45455ae425c83e3b8ed8a67a14261606-am1.d.aa.online-metrix.net*", (*ii*) "*5b584d886b0f49f795209d5763d8c078.events.ubembed.com*", (*iii*) "*y2tyfol9hiuw5hzwe2hnusbzm1qz51545315830.nuid.imrworldwide.com*".

single network; the analysis requires *aramis* security monitoring system. The proposed framework employs a machine learning module which provides a behavioral analysis and specific anomaly indicators able to encompass the characteristics of a covert channel. The main contribution of this approach is the ability to provide network profiles tailored to the network users, and not to the single query events, hence allowing spotting possible deviations from the normal baseline. Moreover, models are created in an unsupervised mode, thus enabling the identification of zero-days attacks and avoiding the requirement of signatures or heuristics for new variants.

The proposed solution has been evaluated over a test network, with the injection of 8 pcaps associated with 7 different APTs and 1 PoS malware campaign and the network traffic of 5 DNS-tunneling tools and 1 DNS-file-transfer tool; all the malicious variants were detected, while producing a low false-positive rate during the same period. Another important contribution of the proposed method is therefore the capability to detect covert channels generated with a wider variety of malware and tools, compared with the state-of-the-art.

As a future development, we plan to extend the analysis to other protocols, (e.g., HTTP and HTTPS), and to refine the linguistic anomaly indicators by adding more languages or custom dictionaries. Moreover, we intend to broaden the behavioral approach in order to create another level of profiles tailored to groups of network users sharing common behavioral characteristics (e.g., same department, same job).

References

[1] J. Ahmed, H. H. Gharakheili, Q. Raza, C. Russell, and V. Sivaraman. Real-time detection of dns exfiltration and tunneling from enterprise networks. *Proc. IFIP/IEEE IM. Washington DC, USA (April 2019)*, 2018.

[2] M. Aiello, M. Mongelli, and G. Papaleo. Supervised learning approaches with majority voting for dns tunneling detection. In *International Joint Conference SOCO14-CISIS14-ICEUTE14*, pages 463–472. Springer, 2014.

[3] M. Aiello, M. Mongelli, and G. Papaleo. Dns tunneling detection through statistical fingerprints of protocol messages and machine learning. *International Journal of Communication Systems*, 28(14):1987–2002, 2015.

[4] Alexa. [Online]. Available: http://www.alexa.com.

[5] S. Aryan, H. Aryan, and J. A. Halderman. Internet censorship in iran: A first look. In *Presented as part of the 3rd USENIX Workshop on Free and Open Communications on the Internet*. USENIX, 2013.

[6] BackBox. [Online]. Available: https://www.backbox.org.

[7] F. Bisio, S. Saeli, P. Lombardo, D. Bernardi, A. Perotti, and D. Massa. Real-time behavioral dga detection through machine learning. In *International Carnahan Conference on Security Technology (ICCST)*, pages 1–6. IEEE, 2017.

[8] BlackArch Linux. [Online]. Available: https://www.blackarch.org.

[9] K. Born and D. Gustafson. Detecting dns tunnels using character frequency analysis. *arXiv preprint arXiv:1004.4358*, 2010.

[10] Y. Bubnov. Dns tunneling detection using feedforward neural network. *European Journal of Engineering Research and Science*, 3(11):16–19, 2018.

[11] P. M. Bureau and C. Dietrich. Hiding in plain sight advances in malware covert communication channels. Black Hat, 2015.

[12] P. Butler, K. Xu, and D. D. Yao. Quantitatively analyzing stealthy communication channels. In *International Conference on Applied Cryptography and Network Security*, pages 238–254. Springer, 2011.

[13] S. Cabuk, C. E. Brodley, and C. Shields. Ip covert channel detection. *ACM Transactions on Information and System Security (TISSEC)*, 12(4):22, 2009.

[14] Cyber intelligence report. techreport, ClearSky, 2017.

[15] A. Das, M.-Y. Shen, M. Shashanka, and J. Wang. Detection of exfiltration and tunneling over dns. In *Machine Learning and Applications (ICMLA), 2017 16th IEEE International Conference on*, pages 737–742. IEEE, 2017.

[16] DET (extensible) Data Exfiltration Toolkit. [Online]. Available: https://github.com/leonjza/dnsfilexfer.

[17] C. Diaz and F. J. Gomez. Dns: A botnet dialect. RootedCON, 2012.

[18] dns2tcp. [Online]. Available: https://github.com/alex-sector/dns2tcp.

[19] DNScapy. [Online]. Available: https://code.google.com/archive/p/dnscapy.

[20] dnscat2. [Online]. Available: https://github.com/iagox86/dnscat2.

[21] dnsfilexfer - File transfers via DNS. [Online]. Available: https://github.com/leonjza/dnsfilexfer.

[22] W. Ellens, P. Żuraniewski, A. Sperotto, H. Schotanus, M. Mandjes, and E. Meeuwissen. Flow-based detection of dns tunnels. In *IFIP International Conference on Autonomous Infrastructure, Management and Security*, pages 124–135. Springer, 2013.

[23] R. Falcone and B. Lee. Oilrig uses ismdoor variant; possibly linked to greenbug threat group. [Online]. Available: https://unit42.paloaltonetworks.com/unit42-oilrig-uses-ismdoor-variant-possibly-linked-greenbug-threat-group.

[24] G. Farnham and A. Atlasis. Detecting dns tunneling. *SANS Institute*, 2013.

[25] Forbes. [Online]. Available: http://www.forbes.com.

[26] A. Giani, V. H. Berk, and G. V. Cybenko. Data exfiltration and covert channels. In *Sensors, and Command, Control, Communications, and Intelligence (C3I) Technologies for Homeland Security and Homeland Defense V*, volume 6201, page 620103. International Society for Optics and Photonics, 2006.

[27] M. F. Grady and F. Parisi. *The law and economics of cybersecurity*. Cambridge University Press, 2005.

[28] T. Grance, K. Kent, and B. Kim. Computer security incident handling guide. *NIST Special Publication*, 800:61, 2004.

[29] J. Grunzweig, M. Scott, and B. Lee. New wekby attacks use dns requests as command and control mechanism. [Online]. Available: https://unit42.paloaltonetworks.com/unit42-new-wekby-attacks-use-dns-requests-as-command-and-control-mechanism.

[30] I. Homem, P. Papapetrou, and DosisSpyridon. Entropy-based prediction of network protocols in the forensic analysis of dns tunnels. *arXiv preprint arXiv:1709.06363*, 2017.

[31] Hybrid Analysis. [Online]. Available: https://www.hybrid-analysis.com.

[32] iodine. [Online]. Available: http://code.kryo.se/iodine.

[33] Joe Sandbox Cloud. [Online]. Available: https://www.joesandbox.com.

[34] Kali Linux. [Online]. Available: https://www.kali.org.

[35] A. M. Kara, H. Binsalleeh, M. Mannan, A. Youssef, and M. Debbabi. Detection of malicious payload distribution channels in dns. In *2014 IEEE International Conference on Communications (ICC)*, pages 853–858. IEEE, 2014.

[36] S. S. Keerthi and C.-J. Lin. Asymptotic behaviors of support vector machines with gaussian kernel. *Neural computation*, 15(7):1667–1689, 2003.

[37] S. Khattak. Characterization of Internet censorship from multiple perspectives. Technical Report UCAM-CL-TR-897, University of Cambridge, Computer Laboratory, Jan. 2017.

[38] M. Korolov. Cyber security review. *Treasury & Risk*, 2012.

[39] J. Liu, S. Li, Y. Zhang, J. Xiao, P. Chang, and C. Peng. Detecting dns tunnel through binary-classification based on behavior features. In *Trustcom/BigDataSE/ICESS, 2017 IEEE*, pages 339–346. IEEE, 2017.

[40] P. Lombardo, S. Saeli, F. Bisio, D. Bernardi, and D. Massa. Fast flux service network detection via data mining on passive dns traffic. In *International Conference on Information Security*, pages 463–480. Springer, 2018.

[41] S. Marchal, J. François, C. Wagner, R. State, A. Dulaunoy, T. Engel, and O. Festor. DNSSM: A large scale passive DNS security monitoring framework. In *2012 IEEE Network Operations and Management Symposium*, pages 988–993. IEEE, 2012.

[42] W. Mercer and P. Rascagneres. DNSpionage campaign targets middle east. [Online]. Available: https://blog.talosintelligence.com/2018/11/dnspionage-campaign-targets-middle-east.html.

[43] A. Nadler. Threat intelligence insights dns-based data exfiltration in the wild. RSAConference.

[44] A. Nadler, A. Aminov, and A. Shabtai. Detection of malicious and low throughput data exfiltration over the dns protocol. *Computers & Security*, 80:36–53, 2019.

[45] L. Oneto. Model selection and error estimation without the agonizing pain. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8(4):e1252, 2018.

[46] OzymanDNS. [Online]. Available: https://dankaminsky.com/2004/07/29/51.

[47] V. Paxson, M. Christodorescu, M. Javed, J. Rao, R. Sailer, D. L. Schales, M. Stoecklin, K. Thomas, W. Venema, and N. Weaver. Practical comprehensive bounds on surreptitious communication over {DNS}. In *Presented as part of the 22nd {USENIX} Security Symposium ({USENIX} Security 13)*, pages 17–32, 2013.

[48] C. Qi, X. Chen, C. Xu, J. Shi, and P. Liu. A bigram based real time dns tunnel detection approach. *Procedia Computer Science*, 17:852–860, 2013.

[49] Reverse.it. [Online]. Available: https://www.reverse.it.

[50] P. Satam, H. Alipour, Y. B. Al-Nashif, and S. Hariri. Anomaly behavior analysis of dns protocol. *J. Internet Serv. Inf. Secur.*, 5(4):85–97, 2015.

[51] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, and J. C. Platt. Support vector method for novelty detection. In *Advances in neural information processing systems*, pages 582–588, 2000.

[52] P. L. Shrestha, M. Hempel, F. Rezaei, and H. Sharif. A support vector machine-based framework for detection of covert timing channels. *IEEE Transactions on Dependable and Secure Computing*, 13(2):274–283, 2016.

[53] A. Shulmin and S. Yunakovsky. Use of dns tunneling for cnc communications. [Online]. Available: https://securelist.com/use-of-dns-tunneling-for-cc-communications/78203.

[54] L. Swersky, H. O. Marques, J. Sander, R. J. Campello, and A. Zimek. On the evaluation of outlier detection and one-class classification methods. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10. IEEE, 2016.

[55] R. W. Taylor, E. J. Fritsch, and J. Liederbach. *Digital crime and digital terrorism*. Prentice Hall Press, 2014.

[56] Threat Spotlight: Inside UDPoS Malware. [Online]. Available: https://threatvector.cylance.com/en_us/home/threat-spotlight-inside-udpos-malware.html.

[57] Tunneling under the sands; possibly linked to greenbug threat group. [Online]. Available: https://asert.arbornetworks.com/tunneling-under-the-sands.

[58] J. Warner and S. Hinck. Threat Spotlight: Inside UDPoS Malware. [Online]. Available: https://www.icebrg.io/blog/footprints-of-fin7-tracking-actor-pattern.

[59] W. E. Winkler. String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage. 1990.

[60] T. Yadav and R. A. Mallari. Technical aspects of cyber kill chain. *arXiv preprint arXiv:1606.03184*, 2016.

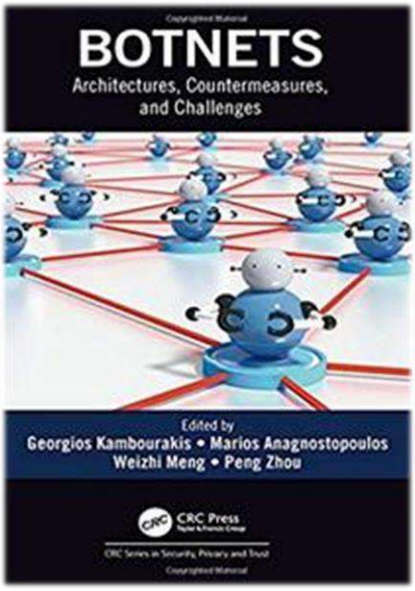
[61] Your-freedom. [Online]. Available: https://www.your-freedom.net.

[62] S. Zander, G. Armitage, and P. Branch. A survey of covert channels and countermeasures in computer network protocols. *IEEE Communications Surveys & Tutorials*, 9(3):44–57, 2007.

BOTNETS: Architectures, Countermeasures and Challenges

GEORGIOS KAMBOURAKIS, MARIOS ANAGNOSTOPOULOS, WEIZHI MENG, PENG ZHOU

Published by CRC PRESS



What is BOTNETS: Architectures, Countermeasures and Challenges?

The book "**Botnets: Architectures, Countermeasures and Challenges**", published by **CRC Press** and edited by G. Kambourakis, M. Anagnostopoulos, W. Meng and P. Zhou provides solid, state-of-the-art **contributions** from both **scientists** and **practitioners** working on **botnet detection and analysis**, including botnet economics. It presents original **theoretical and empirical chapters** dealing with both **offensive and defensive aspects** in this field. Chapters address fundamental theory, current trends and techniques for evading detection, as well as practical experiences concerning **detection and defensive strategies** for the botnet ecosystem, and include surveys, **simulations**, practical **results**, and **case studies**.

Our contribution

Our team contributed to the book with an article titled "***Domain Generation Algorithm Detection Techniques through Network Analysis and Machine Learning***". In this article, we focus on supervised or signature-based approaches and explained their possible limitations. We then discuss the unsupervised techniques, usually retrieved by collecting the DNS traffic of a single network. Eventually, an effective DGA detection algorithm based on a single network monitoring is presented. The proposed approach consists of two steps: the first step involves the detection of a bot looking for the C&C and thus querying many automatically generated domains. The second phase consists on the analysis of the resolved DNS requests in the same time interval. The linguistic and semantic features of the collected unresolved and resolved domains are then extracted in order to cluster them and identify the specific bot. Finally, clusters are analyzed in order to reduce false positives.

Chapter 5

Domain Generation Algorithm Detection Techniques through Network Analysis and Machine Learning

Federica Bisio, Salvatore Saeli and Danilo Massa
aizoOn, Strada del Lionetto, Torino, Italy

Contents

During the last years, the structure and organization of botnets have become more and more challenging. In this context, the role of domain generation algorithms (DGAs) has been crucial to improve the resiliency of communication between bots and command and control (C&C) infrastructure. In fact, these techniques allow botnet controllers to become evasive and potentially avoid detection. In order to efficiently detect these kinds of threats, specific methods have to be implemented. In this context, a number of different approaches to DGA detection have been proposed in state-of-the-art, but DNS-based analysis has resulted to be one of the most appropriate to obtain good results even in near real-time analysis conditions, since it only requires the processing of a small part of the network traffic. For this reason, many recent works focused on automatically recognizing DGA within DNS traffic, whenever occurring.

In this chapter, we will first focus on supervised or signature-based approaches and explain their possible limitations; then, we will discuss the unsupervised techniques, usually retrieved by collecting the DNS traffic of a single network. Eventually, an effective DGA detection algorithm based on a single network monitoring will be presented. The proposed approach consists of two steps: the first step involves the detection of a bot looking for the C&C and thus querying many automatically generated domains. The second phase consists on the analysis of the resolved DNS requests in the same time interval. The linguistic and semantic features of the collected unresolved and resolved domains are then extracted in order to cluster them and identify the specific bot. Finally, clusters are analyzed in order to reduce false positives.

5.1 Introduction

Cybercrime constitutes one of the most serious threats to the current society, with huge consequences on both companies or organizations and single individuals

[1–5]. During the last years, a key role in cybercrime has been played by botnets [6–8], defined as networks of compromised computers (popularly referred to as *zombies* or *bots*), which are controlled by a remote attacker (popularly referred to as a *bot herder*) through specific C&C channels. Among the various threats, DGA-based attacks have recently become a crucial issue to guarantee the success of a botnet, since they allow the improvement of the resiliency of communication between bots and C&C infrastructure.

In fact, the strength of the botnet resides in its highly distributed and highly changeable network, in order to make the tracing and the recovery of all the infected components very difficult, and therefore allowing for the spreading of a wide range of malicious and illegal activities such as ransomwares, exploit kits, or banking trojans [9–13].

In botnets, information can be exchanged by the bot herder and bots using different protocols; for example, peer-to-peer (P2P)-based botnets possess a more robust C&C structure that is difficult to detect and take down, but they are typically harder to implement and maintain. Many attackers try to combine the simplicity of centralized C&Cs with the robustness of P2P-based structures by employing HTTP botnets that locate their C&C servers through the dynamic generation of domains using a DGA, also known as domain flux.

This technique is based on the following steps: first, each bot uses a precalculated seed value known to the bot herder (e.g., the current date) to automatically generate hundreds or thousands of pseudo-random domain names that represent candidate C&C domains. The bot then sends DNS queries until it connects to the IP address associated to a resolved domain. The key advantage of this strategy is that even though one or more C&C domain names or IP addresses are identified and recovered, the bots will query the next set of automatically generated domains and they will eventually get the IP address of a relocated C&C server. In order to obtain a good level of flexibility and a resilient communication channel between bots and C&C, DGAs represent a widely employed technique in botnet control [8,14–20]. Therefore, DGA detection is a task of crucial importance in cyber security.

In this chapter, we will provide an exhaustive overview of state-of-the-art DGA detection methods. Among the number of different approaches, DNS-based analysis is one of the most appropriate to obtain quick responses, since it does not need file dumps and requires only the analysis of a small part of the network traffic (in particular, it can ignore packets' payloads).

Elaborately, there are three main reasons to detect DGA botnets using DNS traces. First, DNS queries are necessary to look up the IP addresses of C&C domains. Second, focusing on a relatively small amount of traffic helps to improve performance, making it possible to detect bots in real time. Third, since bots detection is possible by using only DNS traces when C&C

domains are searched, it might be possible to stop attacks even before they happen.

For these reasons, many recent works focused on automatically recognizing DGA within DNS traffic, whenever occurring. Many efforts have been made to employ supervised or signature-based approaches [21], but these have obtained limited results in the highly dynamic DGA environment. Therefore, some works have applied unsupervised techniques on DNS traffic data provided by some internet service providers [22–25] or retrieved by collecting the DNS traffic of a single network [17,19,26].

After reporting the overview of DNS-based DGAs detection techniques, we will report on an effective DGA detection algorithm that analyzes the DNS traffic of a single network in near real time. In this context, the ability to detect an attack in near real time is crucial, as it allows for a quick reaction, and it is the only way to prevent a potentially severe damage to the company that is working inside the network under attack.

The remainder of the chapter is organized as follows. After the main concepts related to DGA are presented in Sections 5.2, Section 5.3 provides an overview of DGA detection techniques with supervised approaches, while Section 5.4 describes the unsupervised ones. Section 5.5 introduces the monitoring platform that contains the DGA detection method, which is the focus of this chapter and which is described thoroughly with the related experimental results. Finally, conclusions are provided in Section 5.6.

5.2 Background

DGA, also defined as domain flux, is a technique often employed by attackers to hide malicious servers and avoid blacklists. With this technique, each bot, using a precalculated seed value known to the bot herder (e.g., the current date), automatically generates hundreds or thousands of pseudo-random domain names that represent candidate C&C domains. At this point, the bot starts sending DNS queries until it connects to the IP address associated to a resolved domain. The main advantage provided by this strategy is that even if one or more C&C domain names or IP addresses are identified and recovered, the bots will query the next set of automatically generated domains and it will eventually get the IP address of a relocated C&C server.

The technique that instead represents the dual approach employed by attackers is defined as IP flux or fast flux. In fact, a common practice for bot herders is to organize their bots in fast flux service networks (FFSNs): some bots, chosen from a pool of controlled machines, are used as front-end proxies that relay data between a (possibly unaware) user and a protected hidden server. The technique behind these structures is the fast flux, i.e., the rapid and repeated changing of an internet host and/or name server resource record in a DNS zone, resulting in

rapid changes of the IP addresses to which the domain resolves. FFSNs make the tracing and the recovery of all the infected components extremely difficult.

Domains generated by an algorithm are usually pseudo-random domains, sharing at least some common linguistic attributes. It is known however [17] that some modern DGAs employ English dictionary words with little modifica- tions. Therefore, it is usually possible to find common patterns able to character- ize a specific C&C connection and define the behavior of a particular bot.

More specifically, different types of domain layouts can be distinguished:

- Alphabetic or alphanumeric: the characters of the domain are pseudo-random characters extracted from a distribution respectively not containing or containing numbers.
- Dictionary-based: the characters of the domain build words extracted from a dictionary.

In both cases, domains generated by the algorithm may have fixed or variable length.

The following botnets, studied by state-of-the-art works, employ DGAs in order to avoid detection. Some examples are:

- PushDO [27], also known as Pandex or Cutwail, that employs an alphabetic layout of fixed length.
- Kraken [28], also known as Bobax or Oderoor, which employs an alphabetic layout of variable length.
- Necurs [29] that employs an alphabetic layout of variable length.All these variants will be taken into consideration in the experimental evaluation section.

5.3 DGA Detection with Supervised Approaches

Botnets usually rely on DNS to support an agile connection to the C&C. A simple yet effective way to disrupt them is to *blacklist* malicious domains or to add a filtering rule in a firewall or network intrusion detection system.

In an attempt to evade domain name blacklisting, attackers may employ DNS agility. A common example involves the generation of thousands of randomly generated domains with dozens of A records or NS records, or domains used for only a few hours of a botnet's lifetime. Ref. [21] proposes *Notos* to passively analyze DNS query data inside a network. This system is based on the assumption that a malicious use of DNS has unique characteristics that can be distinguished from legitimate DNS services. *Notos* hence builds models of known legitimate domains and malicious domains. In particular, historical DNS information collected passively from multiple DNS resolvers is collected to build a model of legitimate resources, while information about malicious domain names and IP addresses is obtained from sources such as

spam-traps, honeynets, and malware analysis services. Models are built based on statistical features related to information such as geolocalization, domains structure, and number of connections to malicious sources.

After building the models, the system employs them to compute a reputation score for a new domain indicative of whether the domain is malicious or legitimate.

The authors evaluated *Notos* in a large network with DNS traffic from 1.4 million users: the results show that it is able to detect malicious domains with 96.8% of accuracy and low false positive rate (0.38%) and can identify these domains weeks or even months before they appear in public blacklists.

Even though the results are quite satisfying, one of the main limitations of this system is that it is unable to assign reputation scores for domain names with very little historic (passive DNS) information. Therefore, in this situation it might not be trivial to collect data to build an effective supervised classifier. For example, if an attacker always buys new domain names and new address spaces, *Notos* will not be able to accurately assign a reputation score to the new domains. While in the IPv4 space this is very unlikely to happen due to the impending exhaustion of the available address space, it may represent a huge issue for IPv6.

BotCensor [30] is a framework that employs a two-stage anomaly detection to determine if a host is infected with certain DGA malware. In the first stage, a Markov model is used to identify malicious domains, and in the second stage, the potentially malicious hosts are re-examined with novelty detection algorithms. To validate *BotCensor*, the authors conducted a study using both several public source data and real DNS traces. Even though the obtained results are quite satisfying, this system still possesses some limitations. In fact, if an attacker knows the rationale of the first-stage anomaly detection of *BotCensor*, he or she may use domains that are similar to legitimate ones as DNS mapping objects.

Due to the limitations of the supervised approach, in the next section we will consider unsupervised DNS based approaches, which do not need labeled data.

5.4 DGA Detection with Unsupervised Approaches

In the following paragraphs, we propose an overview of state-of-the-art unsupervised approaches, i.e., approaches that do not require prior knowledge of the DGAs or reverse engineering of malware samples.

5.4.1 A Statistical Approach for DGA Detection

In the work proposed by [24], the distribution of alphanumeric characters as well as bigrams in all the domains that are mapped to the same set of IP addresses is taken into consideration. The authors in fact develop metrics borrowing techniques from signal detection theory and statistical learning, which can detect algorithmically

generated domain names that may be generated via plenty of techniques, e.g., pseudo-random string generation algorithms as well as dictionary-based generators. Specifically, they propose the following metrics to quickly differentiate a set of legitimate domain names from malicious ones: information entropy of the distribution of alphanumerics (unigrams and bigrams) within a group of domains; Jaccard index to compare the set of bigrams between a malicious domain name with good domains; Edit-distance, which measures the number of character changes needed to convert one domain name into another.

Their methodology is based on the fact that current botnets do not use well-formed and pronounceable language words since the likelihood that such a word is already registered at a domain registrar is very high. In turn, this means that algorithmically generated domain names can be expected to exhibit characteristics vastly different from legitimate domain names.

5.4.2 Exposure

Among unsupervised approaches, *EXPOSURE* [22] employs a large-scale, passive DNS analysis technique to detect domains that are involved in malicious activity. Fifteen features are extracted from the DNS traffic in order to characterize different properties of DNS names and the ways they are queried.

The experiments were performed on a large real-world data set consisting of 100 billion DNS requests, and a real-life deployment for two weeks has shown that the approach is scalable and able to automatically identify unknown malicious domains that are misused in a variety of malicious activities, e.g., botnet C&C, spamming, and phishing.

Being able to passively monitor real-time DNS traffic allows to identify malware domains that have not yet been revealed by pre-compiled blacklists. Anyway, the system still possesses some limitations: for example, to evade *EXPOSURE*, an attacker could try to avoid the specific features and behavior looked for inside the DNS traffic. Moreover, the detection rate also depends on the training set. Even if the system is not trained on unknown families of malicious domains, the more malicious domains are fed to the system, the more comprehensive the approach can become.

5.4.3 Phoenix

Phoenix [23] is a system that, in addition to detecting DGA- and non-DGA-generated domains using a combination of string and IP-based features, characterizes the DGAs behind them, by finding groups of DGAs that are representative of the respective botnets. As a result, *Phoenix* can associate previously unknown DGA to these groups, and produce novel knowledge about the evolving behavior of each tracked botnet. *Phoenix* framework is hence based on the following phases: collection

of domains, characterization of the generation algorithms, isolation of groups of domains representing the respective botnets, and production of novel knowledge about the evolving behavior of each tracked botnet.

Phoenix has been evaluated on 1,153,516 domains, including DGA-generated domains from well-known botnets: it correctly distinguished DGA- versus non-DGA-generated domains in 94.8% of the cases, and characterized families of domains that belonged to distinct DGAs, helping in gathering intelligence on suspicious domains to identify the correct botnet.

5.4.4 NetFlow

The technique to detect hosts infected by DGA-malware proposed by [17] is based on *NetFlow*, defined as an aggregation of all packets sent from one source IP and port pair to one destination IP and port pair, over the same protocol. DGA-based malware is identified by means of a statistical approach based on the calculation of the ratio of DNS requests and visited IPs for every host in the local network. The system identifies deviations from this model as potential DGA-performing malware. The approach is based on the fact that malware usually tries to resolve many domains during a small time interval without a corresponding amount of newly visited IPs. Large numbers of domain trials are expected because they lower the chance of generating already existing or blocked domains.

Authors show that this method is able to detect different popular bots belonging to different malware families in a real network of 50,000 users with high accuracy.

5.4.5 BotDigger

BotDigger [19] is a system able to detect DGA-based bots using DNS traffic of a single network without a priori knowledge of the specific DGA, by employing the extraction of a chain of evidence, including quantity, temporal and linguistic evidence.

In particular, quantity evidence means that the number of suspicious second-level domains (2LDs) queried by bots is much more than the one of legitimate hosts. Two temporal evidences are used: (1) the number of suspicious 2LDs queried by a bot suddenly increases when it starts to look for the registered C&C domain; (2) once the bot hits the registered C&C domain, the number of queried suspicious 2LDs decreases. The basis of linguistic evidence relates to the fact that the DGA NXDomains (i.e., non-existent domains) and C&C domains queried by a bot are generated by the same algorithm, thus they share similar linguistic attributes.

Authors evaluated *BotDigger* on two famous botnets (Kraken and Conficker) and showed that *BotDigger* was able to detect all the Kraken bots and 99.8% of Conficker bots. Other DNS traces were used to evaluate false positives obtaining false positive rates between 0.05% and 0.39%.

One limitation of this framework resides in the fact that *BotDigger* may not detect DGA if its time window is too large. Anyway, this has the advantage to force bots to take more time to contact the C&C domains in order not to be discovered. Moreover, the quantity evidence requires that the number of NXDomains queried by a bot is comparable more than legitimate hosts. As a result, *BotDigger* will fail only if the bot is "lucky" enough to query just a very small amount of domains before hitting the C&C.

5.5 An Efficient Near Real-Time DGA Approach Based on a Single Network Monitoring

The proposed DGA detection algorithm has been deployed in *aramis* (Aizoon Research for Advanced Malware Identification System) [31], a network security monitoring platform able to automatically identify a wide range of malware and attacks in near real time, through near real-time monitoring of a single network. *aramis*'s structure can be summarized in four phases:

1. *Collection*: sensors placed in various nodes of the monitored network gather data from its different segments, pre-analyze them in real time, and send the results to a NoSQL database.
2. *Enrichment*: inside the NoSQL database, data is enriched with information coming from the aramis Cloud Service, which collects intelligence from various OSINT (Open Source Intelligence) sources and from internally managed sources. Intelligence data include information about IP, domains, and user agents; input data are checked against these sources in order to block potentially blacklisted events. Some OSINT sources are, for example: Alexa, Alienvault, BlockList, MalwareDomains, SANS, PhishTank, Tor Project.
3. *Analysis*: two kinds of analyses are performed on the stored data: (i) *advanced cybersec analytics* to spot and highlight specific patterns of attacks (i.e., DGAs [32], IP Fluxes [33], Ransomware, Covert Channels), and (ii) a *machine learning engine* that applies machine learning algorithms to compare the actual behavior of each node with the usual one, and spot and signal possible deviations from this behavior.
4. *Visualization*: the results are presented through cognitive dashboards, which are crucial to highlight anomalies.

The *machine learning engine* combines the contributions of two unsupervised machine learning approaches (i.e., no data labeling is required), which are the following:

- Bayesian networks: dependences between variables are expressed in a probabilistic way through a directed acyclic graph (DAG), and the probability of anomaly compared to the graph belonging to the historical data is calculated.

Figure 5.1 aramis's dashboard.

- SVM-one class: anomalies are identified in terms of distance from the region including all the points representing the historical data.

Figure 5.1 shows the main dashboard of the framework.

The following subsection describes the DGA detection approach, embedded in the Analysis module.

5.5.1 DGA Detection Method

The aim of the proposed DGA detection method [32] is the near-real-time identification of domain-flux attacks via the monitoring of a single network. To this purpose, the method comprises several steps of analysis.

aramis's DGA detection method

- Collection of unresolved DNS requests (UNRES): all UNRES requests in a suitable amount of time are collected in order to detect the process of a bot trying to connect with the related C&C. A huge and impacting increase of UNRES in a small amount of time may in fact indicate the tentative of connection with several untrusted automatically generated domains.

(*Continued*)

(Cont.)

aramis's DGA detection method

- Filtering and preprocessing of UNRES: all the queries due to user errors (e.g., typos of popular domains) and system misconfigurations are removed.
- Outlier detection: the hosts producing the highest peaks of UNRES are identified.
- Extraction of resolved DNS requests (RES): RES near the peaks identified in the previous step are collected. In this way, it is possible to detect the moment when a bot stops querying because an existent domain has been hit and a successful connection has been established.
- Domain features extraction: all the collected RES and UNRES are mapped in a feature space able to embed the related linguistic and semantic components.
- Clustering: domains with similar features are grouped together in order to spot common patterns of the specific bot, applying specific unsupervised machine learning algorithms.
- False positives removal: in order to reduce false positives, the level of homogeneity of the clusters is calculated. This allows the distinction between true DGAs (associated with highly homogeneous clusters) from the expected legit unresolved DNS peaks (associated with less homogeneous clusters).

Furthermore, we describe the details of each step.

5.5.1.1 Collection of UNRES

In order to maintain the near-real-time constraint, all the UNRES are continuously downloaded and analyzed. On average, the complete DGA detection algorithm takes 2 seconds to complete.

5.5.1.2 Filtering and Preprocessing of UNRES

The following filters are applied to the retrieved UNRES:

- Requests containing invalid or malformed top level domains (TLDs) are removed. Typically, they are due to typos or user errors.
- Overloaded DNS: DNS queries are sometimes overloaded so to provide anti-spam or anti-malware techniques. In order to reduce noise, the overloaded DNS are removed.
- Local and private domains are removed.

- White list domains (i.e., domains that are known to be trusted) are removed.
- Popular domains are removed. More specifically, three popular domains sources are considered—the top 10,000 domains in the world provided by Alexa [34], the web URLs of the 500 world biggest companies provided by Forbes [35], and the top 100 domains collected inside the network under analysis. In all these cases, the second- and third-level domains of an input domain are extracted and compared with the second- and third-level domains of the list of popular domains; if the Jaro-Winkler distance [36] is below 0.1, the input domain is considered as a misspelling of a popular domain and removed.
- Configuration words: domains containing certain substrings (e.g., words related to network system and structure) are filtered out because they represent congenital network traffic.
- ARPA domains are filtered out, since they are only used for reverse DNS lookup.
- If a TLD is found in the third or higher levels, it is considered as a misconfiguration of the web browser or of the particular application and hence it is removed.
- If an IP address is found in the third or following levels, it is considered as an internal domain and it is removed.

The filtering phase removes the largest part of the initial UNRES; usually just 5–10% of the queries are not filtered out and proceed through the other steps of the algorithm.

5.5.1.3 Outlier Detection

In order to recognize burst in the UNRES traffic, time is discretized and the number of UNRES for each machine in each time interval is considered part of a time series, which is described in terms of six different statistical methods:

- deviation from the expected distribution calculated via
  - Gaussian estimate
  - kernel density estimate: this estimate is a non-parametric way to estimate the probability density function of a random variable; the algorithm allows to calculate the probability to belong to a class, taking into consideration the density of the class around the point under analysis
- arima model [37]: this technique is usually applied to time series data to predict future points in the series (forecasting)
- deviation from the expected behavior calculated on a moving window via

- mean and standard deviation
- median and median absolute deviation
- interquartile range

Each method can be considered as a binary classifier between ordinary points and outliers, and the results of all classifiers are combined with an *ensemble classifier* based on a weighted majority rule, where the chosen weight is proportional to the inverse of the mean number of outliers detected by that method: this means that an alarm reported by a method that often presents alarms has a smaller relevance compared to an alarm presented by a usually cautious method. Ensemble classifiers have been shown to perform typically better than any single classifier [38].

The identification of outliers in the distribution of the number of UNRES hence allows to detect potentially suspicious machines.

5.5.1.4 Extraction of Resolved DNS Requests

Once the suspicious machines are detected, the extraction of the related RES is performed. In particular, all the RES occurring in a time interval τ around the UNRES peaks are collected. The interval τ is set to 20 seconds; this choice represents a trade-off between the need of a large τ to compensate possible delays in the network data collection and the necessity of a small τ in order to avoid casual associations of RES with a cluster of UNRES.

5.5.1.5 Domain Features Extraction

The main idea of this phase is the extraction of the most relevant features of both RES and UNRES in order to find common patterns able to characterize a specific C&C connection. In this way, we are able to perform the subsequent clustering phase and group together domains showing a similar pattern, therefore defining the behavior of a particular bot.

To this purpose, we create a common feature space for RES and UNRES, mapping into an array of numbers the linguistic peculiarities of the domains under analysis. This process is built on the assumption that pseudo-random domains generated by the same algorithm typically share at least some common linguistic attributes, while legitimate domains are not generated by an algo- rithm and, hence, should not show similarities in the domain structure. However, it is known [17] that some modern DGAs employ English diction- aries with little modifications; for this reason both linguistic and nonlinguistic features have been considered.

The extracted features are reported in the following.

Linguistic features for domains mapping

- Number of levels in the domain
- For the second and third levels: distance of the monograms probability distribution from the one of monograms in the English language
- For the second and third levels: distance of the bigrams probability distribution from the one of bigrams in the English language
- Entropy in characters distribution of the second and third levels
- Number of characters of the second and third levels

5.5.1.6 Clustering

Once the domain features are extracted, a k-means clustering [39] is performed on the feature space. The number of clusters N_c is set equal to a fifth of the number of input domains because this was found as the best trade-off between the need of a large N_c in order to obtain highly homogeneous groups and the need of a small N_c to avoid the spread of domains belonging to the same DGA into many different clusters. Moreover, every cluster has an associated homo- geneity value corresponding to the average proximity of the samples of the cluster with the related centroid.

After creating the clusters, malicious clusters have to be recognized; they are identified as follows:

- Clusters formed by both RES and UNRES and where the number of UNRES is higher than the number of RES
- Clusters that contain only UNRES

In both cases, we assign an anomaly indicator A to each malicious cluster proportional to its value of homogeneity. Therefore, A has minimum value A 0 (no anomaly detected) and maximum value A 1 (maximum anomaly detected). The two kinds of clusters contain, respectively, DGAs that eventually contacted a C&C, and DGA attempts that did not find a C&C. Thus, A for the second case is reduced by a corrective factor λ_{fail} 0:8. A is hence defined by the following equation:

$$A = \begin{cases} 1 - d_{centroid} & \text{if C \& C is found} \\ (1 - d_{centroid}) \cdot \lambda_{fail} & \text{if C \& C is not found} \end{cases} \quad (5.1)$$

where $d_{centroid}$ is the distance from the centroid of the related cluster.

5.5.1.7 False Positive Removal

The anomaly indicator of each cluster is rescaled in order to reduce false positives. The effect of this rescaling is to further decrease low values of A (usually associated with false positives), to highlight large values of A and to enhance the differences in the interval $(0:3; 0:75]$, which has been recognized in the training phase as the overlapping region between the most uncertain false positives and true positives.

5.5.2 Experimental Evaluation

The DGA detection algorithm described above was evaluated within two different experimental designs:

- Forty DGA snippets belonging to different malware families (including banker trojans, ransomwares, worms) were used to inject real DGA network traffic into an ad hoc network (*malware lab*, see Table 5.1). The malware families of the DGA snippets cover all the most relevant DGA-attack scenarios (see Table 5.2 for a complete list).
  - The LAN of a real company (described in Table 5.1) was observed for a 15-day-long experimental session.

5.5.2.1 First Experiment

The first round of experiments consisted in 40 DGA snippets belonging to different malware families used to simulate real DGA traffic inside the *malware lab*, which is described in Table 5.1. In order to simulate the successful connection to the C&C, a technique similar to *sinkholing* [16,40] was used: before the injection of the traffic generated by each snippet, a couple of the domains produced by the snippet were

Table 5.1 Network Description

	Real Network	*Malware Lab*
Number of machines	288	269
Number of clients	209	185
Average number of connections	136 k/hour	452 k/hour
Average number of UNRES	791/hour	14 k/hour
Average number of RES	59 k/hour	184 k/hour

registered in the FakeDns of the *malware lab*. Each registered domain was associated to an IP address of a honeypot running a web server.[1]

For each malware, Table 5.2 contains the following information:

- Malware type
- Domain layout, i.e., elementary components of the generated domains [18]
- Domain length (fixed or variable)
- Specific names of the malware; aliases of the malware names are reported in square brackets
- Number of clusters containing resolved DNS requests
- Anomaly indicator *A*

From Table 5.2 it is possible to notice that the proposed DGA detection framework successfully detected all the malware variants with a high anomaly indicator. Moreover, all the malicious RES have been identified, thus giving the possibility to detect all the active C&Cs, which were reported to the appropriate OSINT repositories.

5.5.2.2 Second Experiment

The LAN of a real company was observed for a 15-day-long experimental session, in order to provide a real case test of the proposed solution. We considered 21.5 millions of queries, of which 1650 are related to DGA attacks.

To evaluate the performances, we distinguished between RES and UNRES requests: the RES case represents the riskiest situation, since the complete domain-flux attack took place; in this case, therefore, the first concern is the avoidance of false negatives, while some false positives might be tolerated; on the contrary, the UNRES situation is less risky since it indicates that the potential malware unsuccessfully tried to connect to the C&C and a higher false negative rate might be tolerated.

Results reported a 100% detection accuracy of DGA attacks for both cases. Moreover, during the experimental evaluation the false positive rate resulted equal to zero for the RES case, hence allowing to completely distinguish the real attacks from the normal traffic. Also, for the UNRES case, the false positive rate was kept very low at 0.02%. This rate is comparable with the false positive rate obtained by [19]; however, it is important to underline that the proposed framework has been tested over 40 different malware families, while in [19] just two malware variants were taken into consideration.

1 Besides the DNS registered in the experiment, other domains were resolved, revealing the presence of active C&Cs or *sinkholes*.

9780367191542C05.3D 139 [123–146] 17.7.2019 2:02PM

Table 5.2 Malware Description and Detection Results

Malware type	*Domain layout*	*Domainlength*	*Malware names [aliases]*	*Resolved domains*	*A*
Banking Trojan	Alphabetic	Fixed	Fobber [Tinba v3]	2	0.9841
			Ranbyus	5	0.9842
			Tinba [TinyBanker,Zusy]	6	0.9937
		Variable	Qakbot	2	0.9864
			Ramnit	1	0.8885
			Vawtrak [Neverquest,Snifula]	2	0.9638
	Alphabetic + seed	Fixed	Banjori [MultiBanker 2,BankPatch(er)]	3	0.9955
	Alphanumeric	Fixed	Qadars v3	1	0.9850
			Newgoz [GameoverZeus]	3	0.9926
		Variable	Shiotob	2	0.9615
			ZeusBot	1	0.9731
			Murofet v3 [Licat]	1	0.9859
	Alphanumeric +DDNS	Variable	Corebot	4	0.9847
	Dictionary	Variable	Gozi ISFBa [Ursnif, Snifula,Papras]	2	0.9766
			Gozi ISFBb [Ursnif, Snifula,Papras]	3	0.9776
			Rovnix	3	0.9875
Botnet	Alphabetic	Fixed	PushDO [Pandex, Cutwail]	2	0.9995
	Alphabetic +DDNS	Variable	Kraken v1 [Bobax,Oderoor]	5	0.9834
	Alphabetic	Variable	Necurs	2	0.9664
Exploit kit	Alphabetic	Variable	Blackhole	3	0.9924

(*Continued*)

Table 5.2 (Cont.)

Malware type	*Domain layout*	*Domainlength*	*Malware names [aliases]*	*Resolved domains*	*A*
Ransomware	Alphabetic	Fixed	Cryptolocker	2	0.9984
			Padcrypt	4	0.9908
			DirCrypt	2	0.9784
		Variable	Locky v3	3	0.9738
Trojan horse	Alphabetic	Fixed	Dnschanger [Alureon]	1	0.9959
			Ramdo	3	0.9894
			Simda	3	0.9984
			Sisron [TOMB, Trojan.Scar]	1	0.9807
			Srizbi	1	0.9964
		Variable	Bamital	1	0.9888
			Nymaim	3	0.9643
			Vidro	4	0.9866
	Alphabetic + DDNS	Variable	Symmi	2	0.9816
	Alphanumeric	Fixed	Chinad	1	0.9861
		Variable	Beped	2	0.9822
	Dictionary	Variable	Matsnu	3	0.9897
			Suppobox	1	0.9267
Worm	Alphabetic	Fixed	Tempedreve	2	0.9877
		Variable	Proslikefan	5	0.9957
			Pykspa [Pykse,Skyper,SkypeBot]	5	0.9835

Besides, during the experimental session, a real domain-flux attack, including the final contact with the C&C (RES case), has been completely detected. In fact, the alarms associated with this detection were investigated and led to the discovery of the activity of a banking trojan (VawTrak [41]).

From these results, we can conclude that the proposed method is able to detect potentially infected machines in near real time and with high anomaly indicators, while limiting the false positives at the same time.

5.5.2.3 Results and Discussion

The experimental evaluation led to the discovery of a host infected with the Vawtrak malware. Vawtrak, also known as Neverquest, is born from Gozi, another banking Trojan. There are two known versions of Vawtrak, v1 and v2, which continue to be maintained and to receive updates. Vawtrak also supports the use of additional modules, increasing its versatility and the threat it poses

Table 5.3 DGA domains related to the Vawtrak malware

agifdoc.top	agifdocg.top	agufdir.top	alehnomsuc.top
asarwitdi.top	awoflucgufs.top	canefsarg.top	cegafsergo.top
ciwifla.top	cogefdi.top	cogotducnet.top	conitsuc.top
cuwufsecwet.top	cuwutlecnim.top	edehnumsu.top	edohgimli.top
eduhwemsarw.top	egatlorwe.top	egifdarnot.top	enatluh.top
ewefsihnutl.top	fadicnifleh.top	faducwim.top	falehwi.top
fedurga.top	felucnitdor.top	fesecnit.top	fiduhwomde.top
fisehwif.top	fodurgutdo.top	fosarge.top	fosehwotd.top
fosuhgitl.top	fulehwiml.top	fulirwufs.top	fulocgemsa.top
hanatlahgo.top	hawotseh.top	hewutsohgif.top	higotlerwo.top
hiwafduhw.top	hiwatsuh.top	hogetdoc.top	hogutlacwe.top
honamlecn.top	huwamdahgi.top	iducnofd.top	ilacwatd.top
madacnuts.top	malacgim.top	medurne.top	mesohna.top
midacwims.top	modehgamlo.top	modicgofdor.top	mulehwa.top
musucnits.top	ogefsir.top	osuhnimdocg.top	osuhwimso.top
owamsurw.top	owetlurwoml.top	ranomsuhgaf.top	ronitso.top
runamdohg.top	ruwetlocwem.top	tadernatda.top	talahwumsec.top
talocwumder.top	tedihwutlac.top	telurwimlu.top	tesehniml.top
tiluhwomd.top	tisecnemleh.top	tolehnatla.top	udacnofl.top
udihgotlarn.top	ulacwitde.top	ulahgut.top	ulihnef.top
ulorwumder.top	usirnit.top	usuhgutsa.top	uwiflecnatl.top

once it has infected a host. The most commonly distributed modules enable Vawtrak to steal credentials from various applications installed in the host, provide the attackers with remote access, use the host as a proxy, steal certificates, log the user's keystrokes, and use webinjects.

During the experimental evaluation, Vawtrak produced 116 not resolved requests and 54 resolved requests. Examples of domains used by the DGA are reported in Table 5.3.

5.6 Conclusion

In this chapter, an overview of state-of-the-art DGA detection methods has been provided. Among the number of different approaches, the analysis has been focused on the DNS-based detection techniques. In particular, we have presented state-of-the-art supervised or signature-based approaches and explained their possible limitations; then, we have discussed the unsupervised techniques, with particular focus over an effective DGA detection algorithm based on a single network monitoring.

The proposed approach comprises of two steps: the first step involves the detection of a bot looking for the C&C and thus querying many automatically generated domains. The second phase consists of the analysis of the resolved DNS requests in the same time interval. The linguistic and semantic features of the collected unresolved and resolved domains are then extracted in order to cluster them and identify the specific bot. Finally, clusters are analyzed in order to reduce false positives.

References

[1] Tim Grance, Karen Kent, and Brian Kim. Computer security incident handling guide. *NIST Special Publication*, 800:61, 2004.

[2] Maria Korolov. Cyber security review. *Treasury & Risk*,2012.

[3] Frederic Lemieux. Investigating cyber security threats: Exploring national security and law enforcement perspectives. *2011 Developing Cyber Security Synergy*, page 63, 2011.

[4] Robert W Taylor, Eric J Fritsch, and John Liederbach. Digital crime and digital terrorism. *Prentice Hall Press*, 2014.

[5] Tarun Yadav and Rao Arvind Mallari. Technical aspects of cyber kill chain. *arXiv preprint arXiv:1606.03184*, 2016.

[6] Marios Anagnostopoulos, Georgios Kambourakis, and Stefanos Gritzalis. New facets of mobile botnet: Architecture and evaluation. *International Journal of Information Security*, 15(5):455–473, 2016.

[7] David Dagon, Guofei Gu, Christopher P Lee, and Wenke Lee. A taxonomy of botnet structures. In *Twenty-Third Annual Computer Security Applications Conference, 2007. ACSAC 2007*, pages 325–339. IEEE, 2007.

[8] Christian J Dietrich, Christian Rossow, Felix C Freiling, Herbert Bos, Maarten Van Steen, and Norbert Pohlmann. On botnets that use dns for command and control.

In *2011 Seventh European Conference on Computer Network Defense (EC2ND)*, pages 9–16. IEEE, 2011.

[9] Kamal Alieyan, Ammar ALmomani, Ahmad Manasrah, and Mohammed M Kadhum. A survey of botnet detection based on DNS. *Neural Computing and Applications*, 1–18, 2015.

[10] Giles Hogben, Daniel Plohmann, Elmar Gerhards-Padilla, and Felix Leder. Botnets: Detection, measurement, disinfection and defence. *European Network and Information Security Agency*, 2011.

[11] Elaheh Soltanaghaei and Mehdi Kharrazi. Detection of fast-flux botnets through dns traffic analysis. *Scientia Iranica. Transaction D, Computer Science & Engineering, Electrical*, 22(6):2389, 2015.

[12] Matija Stevanovic and Jens Myrup Pedersen. On the use of machine learning for identifying botnet network traffic. *Journal of Cyber Security and Mobility*, 4(3):1–32, 2016.

[13] Hossein Rouhani Zeidanloo, Mohammad Jorjor Zadeh Shooshtari, Payam Vahdani Amoli, M Safari, and Mazdak Zamani. A taxonomy of botnet detection techniques. In *2010 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT)*, volume 2, pages 158–162. IEEE, 2010.

[14] Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou, Saeed Abu-Nimeh, Wenke Lee, and David Dagon. From throw-away traffic to bots: Detecting the rise of dga-based malware. In *USENIX Security Symposium*, volume 12, 2012.

[15] Aymen Hasan Rashid Al Awadi and Bahari Belaton. Multi-phase irc botnet and botnet behavior detection model. *arXiv preprint arXiv:1501.03241*, 2015.

[16] Thomas Barabosch, Andre Wichmann, Felix Leder, and Elmar Gerhards-Padilla. Automatic extraction of domain name generation algorithms from current malware. In *Proceedings of NATO Symposium IST-111 on Information Assurance and Cyber Defense*, Koblenz, Germany, 2012.

[17] Martin Grill, Ivan Nikolaev, Veronica Valeros, and Martin Rehak. Detecting dga malware using netflow. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 1304–1309. IEEE, 2015.

[18] Aditya K Sood and Sherali Zeadally. A taxonomy of domain-generation algorithms. *IEEE Security & Privacy*, 14(4):46–53, 2016.

[19] Han Zhang, Manaf Gharaibeh, Spiros Thanasoulas, and Christos Papadopoulos. Botdigger: Detecting dga bots in a single network. In *Proceedings of the IEEE International Workshop on Traffic Monitoring and Analysis*, 2016.

[20] Marios Anagnostopoulos, Georgios Kambourakis, Panagiotis Drakatos, Michail Karavolos, Sarantis Kotsilitis, and David KY Yau. Botnet command and control architectures revisited: Tor hidden services and fluxing. In *International Conference on Web Information Systems Engineering*, pages 517–527. Springer, 2017.

[21] Manos Antonakakis, Roberto Perdisci, David Dagon, Wenke Lee, and Nick Feamster. Building a dynamic reputation system for DNS. In *USENIX Security Symposium*, pages 273–290, 2010.

[22] Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. Exposure: Finding malicious domains using passive dns analysis. In *NDSS*, 2011.

[23] Stefano Schiavoni, Federico Maggi, Lorenzo Cavallaro, and Stefano Zanero. Phoenix: Dga-based botnet tracking and intelligence. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 192–211. Springer, 2014.

[24] Sandeep Yadav, Ashwath Kumar Krishna Reddy, Narasimha Reddy, and Supranamaya Ranjan. Detecting algorithmically generated malicious domain names. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 48–61, ACM, 2010.

[25] Sandeep Yadav and Narasimha Reddy. Winning with dns failures: Strategies for faster botnet detection. *Security and Privacy in Communication Networks*, pages 446–459, 2012.

[26] Miranda Mowbray and Josiah Hagen. Finding domain-generation algorithms by looking at length distribution. In *2014 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 395–400. IEEE, 2014.

[27] Report about PushDO botnet. https://threatpost.com/pushdo-malware-resurfaces-with-dga-capabilities/100652/.

[28] Report about Kraken botnet. https://johannesbader.ch/2015/12/krakens-two-domain-generation-algorithms/.

[29] Report about Necurs botnet. https://securityintelligence.com/the-necurs-botnet-a-pandoras-box-of-malicious-spam/.

[30] Biao Qi, Jianguo Jiang, Zhixin Shi, Rui Mao, and Qiwen Wang. Botcensor: Detecting dga-based botnet using two-stage anomaly detection. In *2018 17th IEEE International Conference on Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 754–762, IEEE, 2018.

[31] Aramis security monitoring platform. https://aramisec.com/platform.

[32] Federica Bisio, Salvatore Saeli, Pierangelo Lombardo, Davide Bernardi, Alan Perotti, and Danilo Massa. Real-time behavioral dga detection through machine learning. In *2017 International Carnahan Conference on Security Technology (ICCST)*, pages 1–6, IEEE, 2017.

[33] Pierangelo Lombardo, Salvatore Saeli, Federica Bisio, Davide Bernardi, and Danilo Massa. Fast flux service network detection via data mining on passive DNS traffic. In *International Conference on Information Security*, pages 463–480, Springer, 2018.

[34] Top 10000 domains in the world. www.alexa.com.

[35] Top 500 companies in the world. www.forbes.com.

[36] William E Winkler. String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage. 1990.

[37] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. Time series analysis: Forecasting and control. *John Wiley & Sons*, 2015.

[38] Thomas G Dietterich. Ensemble methods in machine learning. *Multiple Classifier Systems*, 1857:1–15, 2000.

[39] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28 (1):100–108, 1979.

[40] Felix Leder, Tillmann Werner, and Peter Martini. Proactive botnet countermeasures: An offensive approach. The Virtual Battlefield: Perspectives on Cyber Warfare, 3:211–225, 2009.

[41] Report about Vawtrak malware. www.blueliv.com/downloads/network-insights-into-vawtrak-v2.pdf.

ISC 2018

THE 21ST INFORMATION SECURITY CONFERENCE

Guildford, UK

September 9th – 12th, 2018

Springer

21st Information Security Conference
ISC 2018

What is ISC?

The **Information Security Conference (ISC)** is an annual conference covering research in theory and applications of **Information Security**. The main focuses of the 2018 edition, which took place at the **University of Surrey**, included **software security**, symmetric chipers and cryptanalysis, **data privacy and anonymization**, outsourcing and assisted computing, advanced encryption, privacy-preserving applications, advanced signatures and **network security**.

Our contribution

Our team presented a paper titled "*Fast Flux Service Network Detection via Data Mining on Passive DNS Traffic*", written by our colleagues from the Aramis team.

In this paper, we report on an effective fast flux detection algorithm based on the passive analysis of the DNS traffic of a corporate network. The proposed method is based on the near-real-time identification of different metrics that measure a wide range of fast flux key features; the metrics are combined via a simple but effective mathematical and data mining approach. The proposed solution has been evaluated in a one-month experiment over the LAN of an enterprise network, with the injection of 47 attacks associated with 9 different malware campaigns. All the fast flux domains were detected with a very low false positive rate and the comparison of performance indicators with a state-of-the-art work shows a remarkable improvement. An in-depth active analysis of a list of malicious fast flux domains confirmed the reliability of the metrics used in the proposed algorithm and allowed for the identification of more than 10000 IPs, some of which are likely associated with compromised hosts. These IPs turned out to be part of two notorious botnets, namely Dark Cloud and SandiFlux, to the description of which we therefore contributed.

Fast Flux Service Network Detection via Data Mining on Passive DNS Traffic

Pierangelo Lombardo[1], Salvatore Saeli[1], Federica Bisio[1], Davide Bernardi[1], and Danilo Massa[1]

[1] aizoOn Technology Consulting, Strada del Lionetto 6, 10146 Turin, Italy
[name].[surname]@aizoongroup.com

Abstract. In the last decade, the use of fast flux technique has become established as a common practice to organise botnets in Fast Flux Service Networks (FFSNs), which are platforms able to sustain illegal online services with very high availability. In this paper, we report on an effective fast flux detection algorithm based on the passive analysis of the Domain Name System (DNS) traffic of a corporate network. The proposed method is based on the near-real-time identification of different metrics that measure a wide range of fast flux key features; the metrics are combined via a simple but effective mathematical and data mining approach. The proposed solution has been evaluated in a one-month experiment over an enterprise network, with the injection of pcaps associated with different malware campaigns, that leverage FFSNs and cover a wide variety of attack scenarios. An in-depth analysis of a list of fast flux domains confirmed the reliability of the metrics used in the proposed algorithm and allowed for the identification of many IPs that turned out to be part of two notorious FFSNs, namely Dark Cloud and SandiFlux, to the description of which we therefore contribute. All the fast flux domains were detected with a very low false positive rate; a comparison of performance indicators with previous works show a remarkable improvement.

Keywords: automated security analysis, malware detection, network security, passive traffic analysis, botnet, fast flux

1 Introduction

During the last few years, the number of cyberattacks with relevant financial impact and media coverage has been constantly growing. As a result, many companies and organizations have been reinforcing investment to protect their networks, with a resultant increase in the research on this topic [1].

Over the last two decades, botnets have represented one of the most prominent sources of threats on the internet: they are networks of compromised computers (popularly referred to as zombies or bots), which are controlled by a remote attacker (bot herder). Botnets provide the bot herder with massive resources (bandwidth, storage, processing power), allowing for the implementation

of a wide range of malicious and illegal activities, like spam, distributed denial-of-service attacks, spreading of malware (such as ransomware, exploit kits, banking trojans, etc.) [19, 21–23, 25].

A common practice for bot herders is to organise their bots in Fast Flux Service Networks (FFSNs): some bots, chosen from a pool of controlled machines, are used as front-end proxies that relay data between a (possibly unaware) user and a protected hidden server. The technique behind these structures is the *fast flux*, i.e., the rapid and repeated changing of an internet host and/or name server resource record in a Domain Name System (DNS) zone, resulting in rapid changes of the IP addresses to which the domain resolves. FFSNs make the tracing and the recovery of all the infected components extremely difficult, thus allowing for a very high availability for illegal online services related to phishing, dumps stores, and distribution of ransomware, info stealers, and click fraud [24, 29, 32, 36, 37, 39].

FFSNs have been known to cybersecurity experts for more than one decade [25, 36], but in the last few years it has been obtaining a spotlight [20, 21, 23, 27, 35, 38]. The renewed interest is related to the studies of large botnets (e.g., Dark Cloud, also known as Zbot network, and the most recent SandiFlux) which make massive usage of fast flux [2, 24, 28]. The standard approach to FFSNs detection is via the so-called *active* DNS analysis, i.e., by actively querying some domains and by collecting and analysing the answers: this strategy has been widely explored and allows for extensive analyses of botnets [20, 25–30, 32, 33].

Instead, the algorithm described in the present work relies on *passive* analysis of the DNS traffic of a single network: it detects the fast flux domains without interaction with the network traffic, thus making the algorithm completely transparent inside and outside the monitored network; in particular, it cannot be uncovered by the attackers, who often control the authoritative name servers responsible for responding to DNS queries about their fast flux domains [34]. The proposed detection approach has been evaluated in a 30-day-long experimental session over the network described in Sect. 5. The performance is much higher compared to a state-of-the-art analogous method [37]. Moreover, the analysis was performed near-real-time: the average execution time of the algorithm was 25 seconds, while the average time between two subsequent runs of the algorithm was 3 minutes (see Sect. 5 for more details).

As an additional test of the proposed approach, we examined the IPs — collected via active DNS analysis — associated with a list of fast flux domains gathered from [3–6]. This investigation confirmed the reliability of the metrics used in the fast flux detection method proposed herein and allowed for the identification of more than 9000 IPs, likely to be associated with compromised hosts, which turned out to be part of two notorious botnets, namely Dark Cloud and SandiFlux, to the description of which we therefore contribute.

The paper is structured as follows. In Sect. 2, we discuss the most relevant features of FFSNs, with an outline of related works. In Sect. 3, we briefly describe *aramis*, the monitoring platform that contains the fast flux detection method

which is the focus of this paper and which is described thoroughly in Sect. 4. Section 5 comprises a detailed discussion of the experimental results of the test of the proposed algorithm, while Sect. 6 contains further investigations on the FFSNs underlying some fast flux domains. Finally, we discuss possible future developments in Sect. 7.

2 Background and Related Work

One of the first works providing an overview of the fast flux attacks was the Honeynet project [36]. In order to explain hidden operations executed by botnets, authors gave examples of both single and double fast flux mechanisms: while the first rapidly changes the A records of domains, the latter frequently changes both the A records and the NS records of a domain. The interested reader can find a review and a classification of fast flux attacks in [39].

Content Delivery Network (CDN) and Round-Robin DNS (RRDNS) are legitimate techniques which are used by large websites to distribute the load of incoming requests to several servers. The response to a DNS query is evaluated by an algorithm which chooses a pool of IPs from a large list of available servers whose number can be of the order of thousands (see Sect. 6 for some examples). As a result, the behaviour in terms of DNS traffic is very similar to the one of a FFSN, and indeed CDNs and RRDNSs represent the typical false positives in fast flux detection algorithms [25, 29, 37].

A large number of approaches have been proposed to detect FFSNs and to distinguish them from legitimate CDNs and RRDNSs. Most of them rely on active DNS analysis, which allows for the collection of a large number of IPs associated with a domain, thus simplifying the FFSNs detection, but they require the resolutions of domains that may be associated with malicious activities [23, 25, 27, 29, 32]. These methods, despite being appropriate for a deep analysis of FFSNs, have relevant drawbacks in implementations oriented to the monitoring of corporate networks [34, 37].

Some FFSN detection methods based on passive DNS analysis have been proposed. Some of them analyse the DNS traffic of a whole Internet Service Provider (ISP), thus taking in input the DNS traffic generated by many different networks. Perdisci et al. [34], in particular, performed a large-scale passive analysis of DNS traffic. They extract some relevant features from the DNS traffic and classified the domains via a C4.5 decision tree classifier. Berger et al. [21] and Stevanovic et al. [38] proposed two other approaches to analyse the DNS traffic of an ISP. Both methods are based on a tool called DNSMap and classify the bipartite graphs formed by the collected fully qualified domain names and the associated IPs. The first method searches for generic malicious usage of DNS, while the latter focuses on FFSNs.

Soltanaghaei and Kharrazi [37], finally, proposed a method for passive DNS analysis of a network which requires a history for each domain to be evaluated and achieved 94.44% detection rate and 0.001% false positive rate in their best experiment. Our algorithm employs a similar approach, but, with a more careful

choice of the metrics achieves better results, while performing a near-real-time analysis (see Sects. 4 and 5 for details).

3 aramis

The proposed fast flux detection technique is included in a commercially available network security monitoring platform called *aramis* (Aizoon Research for Advanced Malware Identification System) [7, 22]. This software automatically identifies different types of malware and attacks in near-real-time, it is provided with dedicated hardware[1], and its structure can be outlined in four phases:

1. Collection: sensors located in different nodes of the network gather data , preprocess them in real-time and send the results to a NoSQL database.
2. Enrichment: data are enriched in the NoSQL database using the information obtained from the aramis Cloud Service, which collects intelligence from various OSINT sources and from internally managed sources.
3. Analysis: stored data are processed by means of two types of analysis: (i) advanced cybersecurity analytics which highlight specific patterns of attacks, among which DGAs [22] and fast flux, and (ii) a machine learning engine which spots deviations from the usual behaviour of each node of the network.
4. Visualization: results are presented in ad hoc dashboards to show and highlight anomalies.

The cycle of the four phases restarts after a time Δt which slightly depends on the traffic flow analysed and amounts to 182 ± 36 seconds on the network described in Sect. 5. A time Δt of this magnitude is the best trade-off between the near-real-time requirement and the need of a large amount of data in order to have statistically significant results.

4 Detection Method

The aim of the proposed detection method is the near-real-time identification of malicious fast flux via the passive monitoring of the DNS traffic of a single network. To this purpose, the method is composed of three steps of analysis. (i) Filtering: queries which are known to be non-malicious (e.g., popular domains, known CDNs, local domains, etc.) are removed. (ii) Metrics identification: some key indicators are calculated over the queries remaining after filters. (iii) Identification: the metrics are used to identify malicious fast flux among the queries.

The three steps have been constructed by combining information on the FFSNs — acquired from the literature — with a simple but effective mathematical and data mining approach. The parameters of the model have been estimated over a *validation set* formed by 30-days of DNS traffic captured from the network described in Table 1, and by 12 pcaps associated with different malware campaigns that leverage FFSNs, collected from the public repository [8].

[1] E5-2690 2.9GHz x 2 (2 sockets x 16 cores) 16 x 8GB RAM, 1.1TB HDD

Table 1. Validation-network description

	30-days total	**one-hour average**
N. of machines	261	-
N. of connections	80 M	111 k
N. of resolved A-type DNS queries	12 M	17 k
N. of unique resolved A-type DNS queries	381 k	527

4.1 Filters

The algorithm receives resolved DNS requests of type A (which return 32-bits IPv4 addresses, in accordance with [9]) collected near-real-time from the monitored network. The first step consists in the application of the filters reported in Table 2 to the retrieved queries.

Table 2. Filters description

Type	**Description**
White list domains	Domains known to be trusted, e.g., the ones associated with crypto currencies (if their use is allowed in the network): the underlying peer-to-peer networks are, in many respects, similar to botnets.
Popular domains	Top 100 domains collected inside the network under analysis, web URLs of the 500 world biggest companies provided by Forbes [10] and top 10000 domains in the world provided by Alexa [11].
Configuration words	Domains containing certain substrings (e.g., related to network system and structure) represent congenital traffic.
Overloaded DNS	In order to provide anti-spam or anti-malware techniques, DNS queries are sometimes overloaded, thus causing possible noise.
Local and corporate domains	These domains represent a high percentages of the legitimate DNS traffic in a corporate network.
CDNs and RRDNSs	These are the most common sources of false positives in fast flux detection algorithms; aramis (see Sect. 3) includes a function (with a structure similar to that of the proposed fast flux detection algorithm) which periodically updates a white list with the main CDNs and RRDNSs detected in the monitored network, thus allowing for a substantial speed up.
Queries with large TTL	According to the literature, malicious fast flux are characterised by a short TTL [25, 32, 37], therefore queries with a TTL larger than 1800 s are filtered.

4.2 Metrics Identification

The DNS requests that survive the filters described in Sect. 4.1 are integrated with the history of the previous 30 days, saved locally. This allows for a more

accurate evaluation of the behaviour of the domains, however an assessment is already possible when the first answer is received. Among the remaining domains there are many new emerging CDNs[2] and, in order to distinguish them from the FFSNs — which is the main challenge in malicious fast flux detection — we identified some key indicators. Some of these indicators can be already evaluated after a single query (we call them *static metrics*), while others need a certain history (*history-based metrics*). The information regarding Autonomous Systems (ASs) and public networks used in the following metrics are retrieved from [12].

Static Metrics. The metrics described in this section are evaluated over all the IPs collected.

Maximum Answer Length. A relevant metric for the detection of malicious fast flux is the number of IPs returned in a single A query. In particular, we consider the maximum m_{al} of such value: a malicious fast flux is believed to typically have a m_{al} larger than a legitimate fast flux [25, 39].

Cumulative Number of IPs. Malicious fast flux typically employ a larger number of IPs (n_{IP}) compared with CDNs, due to the lower reliability of each single node [37].

Cumulative Number of Public Networks. Since the botnet underlying a malicious fast flux contains infected machines which are typically distributed quite randomly in different networks, the same is expected to be true for the IPs retrieved by the related queries [25, 39]. For this reason a malicious fast flux typically has a number of public networks (n_{net}) larger than a legitimate CDN.

Cumulative Number of ASs. For the same reason described above, FFSNs typically have a number of ASs (n_{AS}) larger than legitimate CDNs.

AS-Fraction. The analysis of some preliminary fast flux pcaps revealed that, despite being in general very useful, in some cases the absolute number of AS was not a distinctive feature, while its ratio with the number of IPs was more appropriate. For this reason we defined the metric

$$f_{\mathrm{AS}} = \frac{n_{\mathrm{AS}} - 1}{n_{\mathrm{IP}}}, \tag{1}$$

which quantifies the degree to which the IPs are dispersed in different AS. This quantity takes values from $f_{\mathrm{AS}} = 0$ (when all the IPs are in the same AS) to $f_{\mathrm{AS}} \sim 1$ (when each IP is in a different AS and the number of IPs is large). In order to preserve these properties and to encode the additional information about the typical scales associated with n_{AS} for CDNs and FFSNs respectively, we rescaled f_{AS} as described below. The first rescaling is

$$x \;\longrightarrow\; \theta(n_{\mathrm{AS}} - n_0)\left[1 - e^{-\left(\frac{n_{\mathrm{AS}} - n_0}{s}\right)^2}\right] x, \tag{2}$$

where $x = f_{\mathrm{AS}}$, $\theta(t)$ is the Heaviside step function (i.e., $\theta(t)$ is 1 for positive t and 0 otherwise), s is a scale representing the average number of ASs in a

[2] The filter mentioned in Table 2 detects a CDN only when it has a sufficient history.

typical CDN and n_0 is a threshold for n_{AS} below which the behaviour is not suspicious from the viewpoint of the number of ASs.[3] The rescaling in Eq. 2 reduces f_{AS} when its value is comparable with the n_{AS} expected in a CDN. The second rescaling applies Eq. 2 to the quantity $x = 1 - f_{\mathrm{AS}}$ and reduces it (i.e., increases f_{AS}) when n_{AS} is comparable with that of a typical FFSNs. In this case the scale s represents the average number of ASs in a typical malicious fast flux, while n_0 is a threshold for n_{AS} below which we do not increase f_{AS}.[4]

IP-Dispersion. The analysis of the distribution of the retrieved IPs is another way to understand to which degree the structure underlying FFSN is random and chaotic. We transform the set of the n IPs associated with each query into the corresponding positions in the 32-bits IPv4 address space $x_1, ...x_n$,[5] and we define

$$d_{\mathrm{IP}} = \frac{1}{l_n} \operatorname{median}(\Delta \mathbf{x}), \tag{3}$$

where $\Delta \mathbf{x} = \{x_i - x_{i-1}\}_{i=2}^{n}$, the $\{x_i\}$ have been ordered so that $x_i \geq x_{i-1}$, and l_n is the average distance if the n IPs were uniformly distributed in the whole public IPs address space. The IP-dispersion takes value from $d_{\mathrm{IP}} = 1$ (i.e., when the IPs are uniformely distributed among the whole public IP space) to $d_{\mathrm{IP}} = 0$ (i.e., when the IPs are clearly subdivided into a few clusters of close addresses). A similar idea was used by Nazario et al. [32], who evaluated the average distance among the $\{x_i\}$, but their metric is more sensitive to outliers and it is not normalised in the interval [0,1], which is crucial to combine it with the other metrics, as described in Sect. 4.3. The FFSNs analysis described in Sect. 6 confirmed that the indicator in Eq. 3 is able to catch the key distribution properties of IPs in a FFSN.

History-Based Metrics. The *history* is constructed by subdividing the queries retrieved from the monitored network in subsequent *chunks*: each chunk contains at least 10 queries and spans a time interval of at least one hour; these two conditions are the minimal requirements to make the metric definitions meaningful from a statistical point of view. The metrics described in this section are evaluated only if it is possible to construct at least two chunks.

Change in the set of IPs. It is a common belief that, while a CDN typically returns IPs taken from a stable IP-pool, a malicious fast flux employs the available nodes in the FFSN, which often evolves quickly, and therefore its IP-pool changes from time to time [25, 39]. We defined a metric which measures in a very

[3] We set $s = 2.5$ and $n_0 = 3$; the first is the average n_{AS} for the top 4 largest CDNs detected in the validation set, while the latter is half the minimum of n_{AS} detected for a fast flux in the validation set.

[4] We set $s = 40$ in agreement with Ref. [39], which states that a typical FFSN has a set of IPs distributed among 30–60 ASs, and $n_0 = 5$, which is the maximum number of ASs detected for a CDN in the validation set.

[5] To each IP $n_1.n_2.n_3.n_4$ we associated $x = 256^3\, n_1 + 256^2\, n_2 + 256\, n_3 + n_4$.

simple way the change in the IP-pool:

$$c_{\mathrm{IP}} = \frac{n_{\mathrm{IP}}}{n^{\mathrm{c}}_{\mathrm{IP}}} - 1, \tag{4}$$

where $n^{\mathrm{c}}_{\mathrm{IP}}$ is the number of unique IPs present in the chunk averaged over all chunks, while n_{IP} has been defined in Sect. 4.2. This quantity takes the value $c_{\mathrm{IP}} = 0$ when all the IPs are found in each chunk, i.e., when the IP-pool is stable and it is explored completely in each chunk (and therefore $n^{\mathrm{c}}_{\mathrm{IP}} = n_{\mathrm{IP}}$). On the other hand, if the IP-pool changes substantially from one chunk to the other, the total number of IPs n_{IP} is much larger than the average number of IPs $n^{\mathrm{c}}_{\mathrm{IP}}$ found in a chunk, and therefore c_{IP} becomes large (it is unbounded above). The same considerations apply to all the following metrics.

Change in the Set of Public Networks. While CDNs typically use IPs taken from the same few public networks, malicious fast flux frequently introduce IPs from new networks [25, 39]. We measure the change in the set of public networks by means of $c_{\mathrm{net}} = n_{\mathrm{net}}/n^{c}_{\mathrm{net}} - 1$, where $n^{\mathrm{c}}_{\mathrm{net}}$ is the network-analogous of $n^{\mathrm{c}}_{\mathrm{IP}}$.

Change in the Set of ASs. The generalisation of the previous argument to the next aggregation level brings us to the analysis of the changes in the number of AS involved. We introduce therefore $c_{\mathrm{AS}} = n_{\mathrm{AS}}/n^{c}_{\mathrm{AS}} - 1$, where $n^{\mathrm{c}}_{\mathrm{AS}}$ is the AS-analogous of $n^{\mathrm{c}}_{\mathrm{IP}}$.

Change in the Answer Length. Another relevant indicator is the change in the number of IPs retrieved in each query [25, 39]. We measure this change by means of $c_{\mathrm{al}} = m_{\mathrm{al}}/m^{c}_{\mathrm{al}} - 1$, where $m^{\mathrm{c}}_{\mathrm{al}}$ is the m_{al}-analogous of $n^{\mathrm{c}}_{\mathrm{IP}}$.

4.3 Fast Flux Domains Identification

A preliminary step for fast flux domains identification is the filtering of the queries with $d_{\mathrm{IP}} = 0$, because this removes many false positives with no loss in terms of true positives. The next step is the use of the metrics defined in Sect. 4.2 to discriminate among malicious fast flux and CDN. Instead of using a machine learning 'black box' classifier, we combine the indicators in a controlled way, in order to encode some other domain knowledge and to allow for an easier interpretation of the results. Foremost we aggregate the static and history-based metrics separately, and finally we combine them into a single anomaly indicator A, which can straightforwardly be used to classify the queries between fast flux and legit domains.

Aggregation of the Static Metrics. We normalised the metrics n_{IP}, n_{net}, n_{AS}, and m_{al} in the interval [0,1], so that for all of them the value 0 corresponds to a typical CDN, while 1 corresponds to the expected behaviour of a malicious fast flux. This is achieved by means of a square-exponential scaling of the form

$$x \longrightarrow 1 - e^{-\left(\frac{x-x_0}{s}\right)^2}, \tag{5}$$

where $x_0 = 1$ is the minimum value for the metric before the rescaling, s is different for each metric and represents an intermediate scale between a typical CDN behaviour and a behaviour clearly ascribed to a malicious fast flux.[6] Equation 5 rescales $x = x_0$ (i.e., the smallest possible value for x), $x = s + x_0$ (i.e., a value intermediate between the typical CDN behaviour and the typical malicious behaviour), and $x \gg s$ (i.e., a value much larger than the scale s) to 0, 1/2, and 1 respectively.

After the scaling, all the quantities n_{IP}, n_{net}, n_{AS}, m_{al}, f_{AS}, and d_{IP} are comparable: they take values in the interval [0,1] and for each of them a value close to 0 denotes a typical CDN behaviour, while a value close to 1 indicates a very suspicious behaviour. We combined these indicators with a weighted arithmetic mean in a unique static index[7]

$$A_{\mathrm{stat}} = w_{\mathrm{IP}} n_{\mathrm{IP}} + w_{\mathrm{net}} n_{\mathrm{net}} + w_{\mathrm{AS}} n_{\mathrm{AS}} + w_{\mathrm{al}} m_{\mathrm{al}} + w_f f_{\mathrm{AS}} + w_d d_{\mathrm{IP}}. \quad (6)$$

In order to avoid the evaluation of misleading indicators due to lack of data, the metrics f_{AS} and d_{IP} are evaluated only if a minimum number of IPs is collected, while m_{al} is evaluated only if at least one answer contains more than one IP. When one metric is absent, its value is set to 0 (in the absence of data we apply a sort of 'presumption of innocence', to reduce false positives), its weight in the evaluation of A_{stat} is decreased by a factor 20 (because the innocence assessment is only due to the absence of data), and the other weights are proportionally rescaled so that $\sum_i w_i = 1$.

Aggregation of the History-Based Metrics. As already mentioned, the metrics c_{IP}, c_{net}, c_{AS}, and c_{al} defined in Sect. 4.2 are unbounded above. We normalise them in the interval [0,1] by means of Eq. 5 with $x_0 = 0$ (as the minimum value for these metrics before the rescaling is 0).[8] After the rescaling, all the metrics take values in the interval [0,1] and for each of them a value close to 0 corresponds to a very stable behaviour, while a value close to 1 indicates a behaviour with high variability over time. We combine then in a unique indicator three of the history-based metrics (the fourth, i.e., c_{al} is instead used in Eq. 8) with a weighted arithmetic mean[9]

$$A_{\mathrm{dyn}} = w'_{\mathrm{IP}} c_{\mathrm{IP}} + w'_{\mathrm{net}} c_{\mathrm{net}} + w'_{\mathrm{AS}} c_{\mathrm{AS}}. \quad (7)$$

[6] The values of s were set based on information retrieved from the literature ([39] and references therein) and the validation set. More in detail, we chose $s_{\mathrm{IP}} = 24$, $s_{\mathrm{net}} = 12$, $s_{\mathrm{AS}} = 6$, and $s_{\mathrm{al}} = 10$.

[7] The weights reflect the importance of the corresponding metric in the correct classification in the validation set; the optimal values are $w_{\mathrm{IP}} = w_{\mathrm{net}} = 0.03$, $w_{\mathrm{AS}} = 0.13$, $w_{\mathrm{al}} = 0.09$, $w_f = 0.54$, and $w_d = 0.18$.

[8] The values of s were set based on information retrieved from the literature and the validation set. More in detail, we chose $s_{\mathrm{IP}} = s_{\mathrm{net}} = 1$ and $s_{\mathrm{AS}} = s_{\mathrm{al}} = 0.5$.

[9] The weights reflect the importance of the corresponding metric in the validation set; the optimal values are $w'_{\mathrm{IP}} = 0.07$, $w'_{\mathrm{net}} = 0.23$, and $w'_{\mathrm{AS}} = 0.7$.

Final Aggregation. We combine the indicators A_{stat}, A_{dyn}, and c_{al} into a single anomaly indicator A, which should be used to classify the queries between fast flux and legit domains. In order to reduce false positives, we differentiate on the basis of the quantity f_{AS}, and we define

$$A = \begin{cases} \sum_i w_i A_i & \text{if } f_{\text{AS}} \geq 0.5 \\ \prod_i (A_i)^{w_i} & \text{if } f_{\text{AS}} < 0.5 \end{cases}, \tag{8}$$

where $\{A_i\} = \{A_{\text{stat}}, A_{\text{dyn}}, c_{\text{al}}\}$ and $\{w_i\}$ are the related weights.[10] Analogously to the averages in Eqs. 6 and 7, when one metric is absent, its value A_i is set to 0 (not anomalous), its weigth w_i is decreased by a factor 20, and the other weights are proportionally rescaled so that $\sum_j w_j = 1$.

Note that in Eq. 8 a (weighted) arithmetic mean is used when the AS-fraction is large, while a (weighted) geometric mean is used when the AS-fraction is small; this implies that in the latter case a value close to 0 for one of the indicators A_i gives a stronger penalty to A.

The detection of malicious fast flux has thus been reduced to a very simple one-dimension classification problem: only queries with $A > A_{\text{th}}$ are labeled as fast flux, where the optimal threshold ($A_{\text{th}} = 0.25$) has been found by maximizing the performance on the validation set. In order to increase the readability of the results, we applied a sigmoid-shaped rescaling which maps $A = 0$ and $A = 1$ onto themselves and A_{th} onto 0.5.

5 Experimental Evaluation

The fast flux detection algorithm described in Sect. 4 was evaluated over a test set comprising 30 days of ordinary traffic of the network described in Table 3 with the injection of fast flux traffic which covers all the most relevant fast flux attack scenarios (see Table 4 for a complete list). Note that the test set has been only used to test the performance of the algorithm and not to modify the algorithm or the parameters.

Table 3. Test-network description

	30-days total	**one-hour average**
N. of machines	391	-
N. of client machines	286	-
N. of connections	398 M	552 k
N. of resolved A-type DNS queries	75 M	104 k
N. of unique resolved A-type DNS queries	1.3 M	1.9 k

The fast flux traffic has been injected in the network via 47 pcaps — collected from the public repositories [3–5] — which are associated with 9 different mal-

[10] An optimisation procedure on the validation set produced similar weight for the three quantities: $w_{\text{stat}} = 0.27$, $w_{\text{dyn}} = 0.38$, and $w_{\text{al}} = 0.35$.

ware campaigns. Table 4 provides a brief description of each malware campaign with the following information:

- the category, i.e., the malware type associated with the campaign
- the name of the campaign
- the list of the domains present in each pcap of the campaign, with the anomaly indicator A associated by the algorithm to each of them
- the average value of A for each campaign

Table 4. Malware description

Category	**Campaign**	**Domains (A)**	$\langle A \rangle$
Banking Trojan	ZBOT	miscapoerasun.ws (0.85)	0.85
Banking Trojan	Dreambot	rahmatulahh.at (0.89); ardshinbank.at (0.92)	0.91
Banking Trojan	Ursnif	widmwdndghdk.com (0.90); bnvmcnjghkeht.com (0.85); qqweerr.com (0.85)	0.87
VBA Dropper	Doc Dropper Agent[a]	aassmcncnnc.com (0.90); iiieeejrjrjr.com (0.87); ghmchdkenee.com (0.88)	0.88
Ransomware	Locky	thedarkpvp.net (0.83); nsaflow.info (0.91); mrscrowe.net (0.93); sherylbro.net (0.87); gdiscoun.org (0.90); scottfranch.org (0.90)	0.89
Ransomware	Nymaim	iqbppddvjq.com (0.91); danrnysvp.com (0.91); pmjpdwys.com (0.93); vqmfcxo.com (0.86); gbfeiseis.com (0.91); danrnysvp.com (0.87); iuzngzhl.com (0.97); vpvqskazjvco.com (0.84); jauudedqnm.com (0.93); dtybgsb.com (0.93); tuzhohg.com(0.93); sxrhysqdpx.com (0.86); arlfbqcc.com (0.93)	0.90
Banking Trojan	Zeus Panda	farvictor.co (0.89); fardunkan.co (0.89); bozem.co (0.84); farmacyan.co (0.87); fargugo.co (0.90); manfam.co (0.85)	0.87
Banking Trojan	GOZI ISFB	qdkngijbqnwehiqwrbzudwe.com (0.80); jnossidjfnweqrfew.com (0.90); zxciuniqhweizsds.com (0.86); huwikacjajsneqwe.com (0.92); efoijowufjaowudawd.com (0.92); onlyplacesattributionthe.net (0.90); nvvnfjvnfjcdnj.net (0.86); popoiuiuntnt.net (0.89); zzzzmmmsnsns.net (0.80); popooosneneee.net (0.83); liceindividualshall.net (0.87); roborobonsnsnn.net (0.93)	0.87
Ransomware	GandCrab	zonealarm.bit (0.90)	0.90

[a] Doc.Dropper.Agent-6332127-0 [13]

Table 4 clearly shows that the proposed method successfully detected all the fast flux domains with a high anomaly indicator. In fact the value of A averaged over all campaigns is equal to 0.89.

Table 5. Results

	$A > 0$	$A > 0.5$
True Positives (T_P)	47 (100%)	47 (100%)
False Negatives (F_N)	0 (0%)	0 (0%)
False Positives (F_P)	6 (<0.001%)	4 (<0.001%)

In Table 5 we summarise the performance of the algorithm: in the second column we consider the total number of outputs of the algorithm (i.e., the number of domains with $A > 0$) while in the third column we report the number of outputs labeled as fast flux (i.e., the number of domains with $A > 0.5$). On the rows we report the following quantities

- True Positives rate (T_P): the number of unique fast flux domains detected;
- False Negatives rate (F_N): the number of unique fast flux domains incorrectly labeled as legit;
- False Positives rate (F_P): the number of unique legit domains incorrectly labeled as fast flux.

All rates are given as absolute values and as percentages for each type on the corresponding number of unique domains in input.

A remarkable result is the absence of false negatives: this determines indeed a 100% recall, also known as detection rate, $R = T_P/(T_P + F_N)$. In order to evaluate the algorithm also with a metric that takes into account the false positives rate F_P, we computed the F-score $F = 2\,P\,R/(P + R)$ (where $P = T_P/(T_P + F_P)$), obtaining $F = 95.9\%$.

As a comparison, [37] obtained $R = 94.4\%$ and $F = 89.5\%$ in their best experimental result. We can therefore conclude that the proposed method is able to detect queries to fast flux domains in a corporate network in near-real-time and with high anomaly indicators, limiting false positives at the same time.

6 Fast Flux Service Networks Analysis

As an in-depth analysis of the algorithm described in Sect. 4, we examined the IPs associated to a list of fast flux domains. The IPs were collected via active DNS analysis and precisely with an FFSN-spanner which resolved systematically domains taken from a list of malicious domains; these domains were gathered via a scouting activity from the public repositories [3–6]. With the purpose of hiding the FFSN-spanner activity from the bot herders, we randomized the sequence of the queries and the waiting times among two subsequent queries, while implementing an anonymization technique based on the use of the Tor network. In order to overcome a limitation of the DNSPort resolver [14], which returns only the first answer for domain lookup, we adopted *ttdnsd*, the Tor TCP DNS Daemon. This solution allows for making arbitrary DNS requests by converting any UDP request into a TCP connection, which is given to Tor through the SOCKS

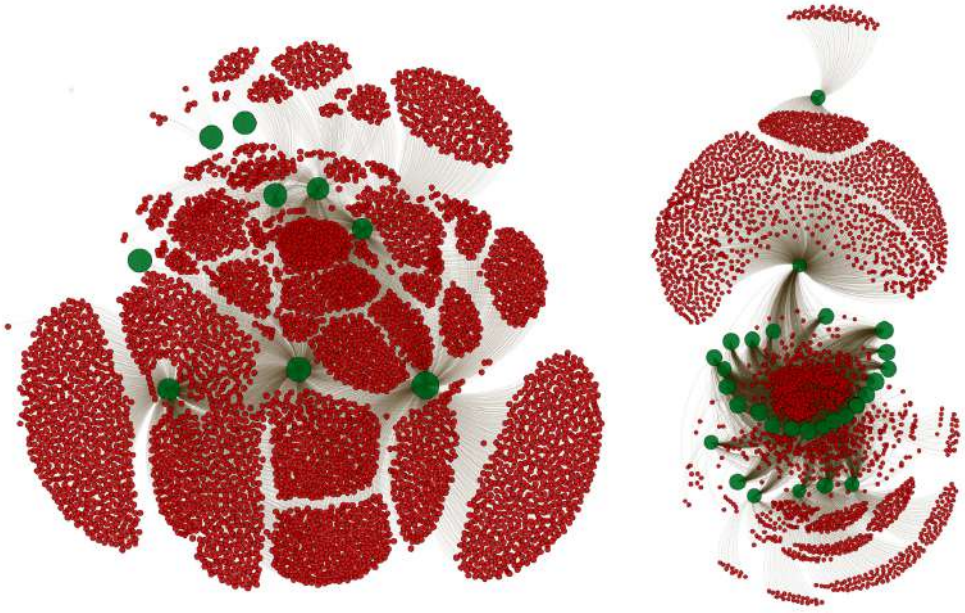


Fig. 1. Bipartite graph representing the IPs (*small red circles*) associated with each domain (*large green circle*). An arc indicates that the IP has been given in answer to a query resolving the domain.

port. The request is then forwarded anonymously through the Tor network and reaches one of the 'open' recursive name servers via the Tor Exit node.

Over the period 09.03.2018–15.04.2018, we collected 9861 IPs (of whom 9049 are public IPs and 812 are reserved IPs) associated with 61 domains, represented in Fig. 1: it can be noted that different domains (*large green circles*) share some IPs (*small red circles*). In Fig. 2 we represent instead the overlap O_{ij} among all the pairs (i, j) of the top 30 domains (for number of collected IPs), defined as

$$O_{ij} = \frac{|X_i \cap X_j|}{|X_i \cup X_j|}, \tag{9}$$

where X_i is the pool of IPs associated with the i-th domain and $|X|$ is the the cardinality of X.

Both Figs. 1 and 2 show a clear subdivision of the domains in two independent clusters. The analysis of the fast flux domains revealed that the clusters correspond to two large FFSNs, namely Dark Cloud (on the right in Figs. 1 and 2) and SandiFlux (on the left in Figs. 1 and 2). Indeed, in the first cluster we recognised domains that are associated with phishing activities, Dumps Stores and malware campaigns (e.g., ZBOT, Dreambot, Ursnif, Locky, and the last GOZI ISFB campaign) that leverage Dark Cloud [15, 16, 24, 28], while in the latter we found domains associated with the Nymaim campaign, which has been

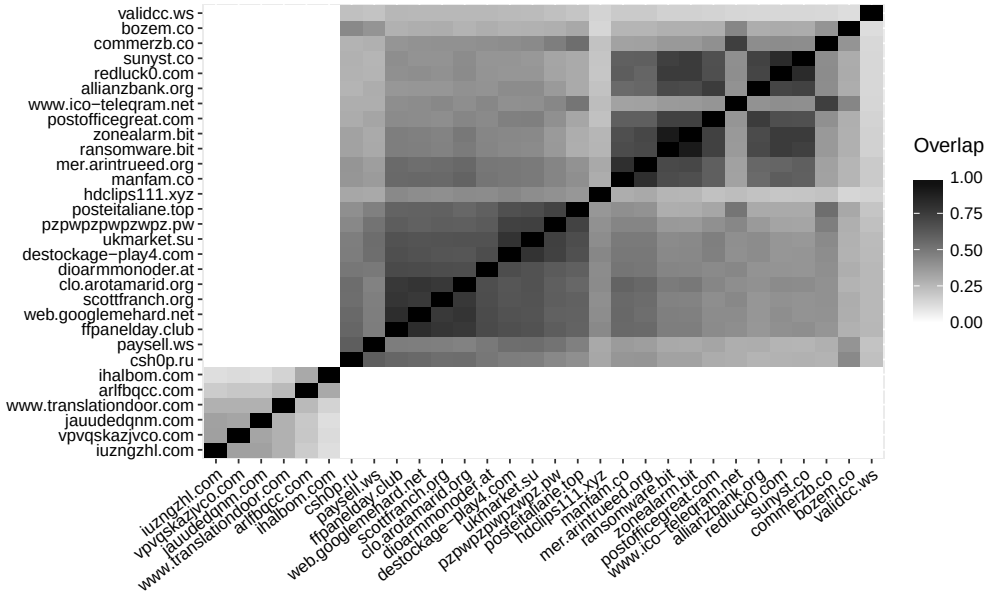


Fig. 2. Overlap representation O_{ij} (defined in Eq. 9) among all the pairs (i, j) of the top 30 domains (for number of retrieved IPs). Darker tones represent larger overlaps.

related with Sandiflux [2]. We also discovered that Doc.Dropper.Agent-6332127-0 [13], Zeus Panda [17], and GandCrab campaigns belong to Dark Cloud, even though the latter is believed to leverage SandiFlux [2].

It is worth noting that the sets of IPs in the two FFSNs that we identified are highly overlapped, but no IP is shared among the two groups. Note also that the clusters that we identify as Dark Cloud and Sandiflux are similar, respectively, to the *Hosting Network* and *C&C Network* described in [28].

The subdivision in two different FFSNs is reflected in the different geolocation of the relative IPs: Fig. 3 shows that, while the IPs retrieved from SandiFlux spread from USA to Europe and Asia, the ones retrieved from Dark Cloud are mainly localised in eastern Europe. In Fig. 4 we further investigate the top 10 countries represented in each botnet: Ukraine, Romania, Bulgaria, and Russia confirmed to be the most represented countries in Dark Cloud [24], while SandiFlux's IPs are found mainly in USA and China. It can be noticed that many IPs associated with the domains of SandiFlux are reserved by Iana [18]. This is probably due to a malicious usage of a practice called domain parking [31] to introduce noise into black lists. Figures 3 and 4 are based on the IP-geoloc tables downloaded from Maxmind [12].

The two FFSNs described above are a good testing ground for the metrics introduced in Sect. 4.2: in Table 6 we report a summary of some of these metrics

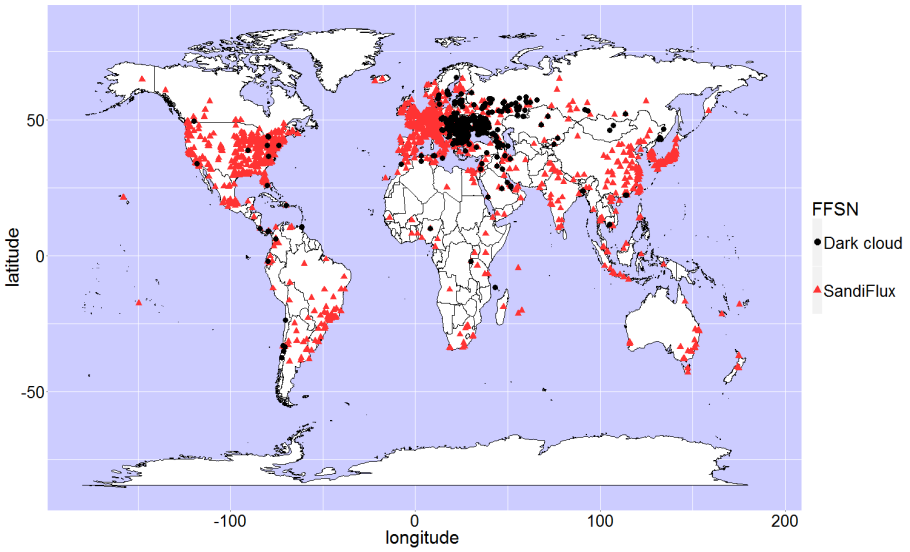


Fig. 3. Geolocation of the IPs retrieved for the two FFSNs

evaluated over the two FFSNs and three large CDNs. Note that two of the CDNs we observed (namely, www.nationalgeographic.it and cdn.wetransfer.net) have a very large number of IPs, making thus n_{IP} a misleading indicator in these cases. This is not a problem for the proposed algorithm, since, as explained in Sect. 4, n_{IP} is used in combination with many other metrics.

Table 6. Summary of some relevant metrics

	n_{IP}	n_{AS}	$n_{\mathrm{IP}}^{\mathrm{resc}}$	$n_{\mathrm{AS}}^{\mathrm{resc}}$	$c_{\mathrm{AS}}^{\mathrm{resc}}$	$f_{\mathrm{AS}}^{\mathrm{resc}}$	d_{IP}
Dark Cloud	2856	354	1	1	1	1	$1.2\ 10^{-3}$
Sandiflux	6203	1517	1	1	1	1	0.69
www.nationalgeographic.it	2478	1	1	0	0	0	$2.6\ 10^{-6}$
cdn.wetransfer.net	2734	1	1	0	0	0	$2.9\ 10^{-6}$
neo4j.com	29	1	0.74	0	0	0	$5.1\ 10^{-4}$

In Fig. 5 we represent the histogram of the frequencies of the first byte in the IP-pool of the two FFSNs and one medium-size CDN. A clear difference between botnets can be noticed. In particular SandiFlux, where the IP-distribution is not so far from a uniform random distribution and the CDN 'imap.gmail.com', where the IP-distribution has a few high peaks. Figure 5 clearly shows that simple indicators as the mean and the variance (represented by the corresponding Gaussian distribution in Fig. 5) do not catch the nature of the distribution, while the metric d_{IP} defined in Eq. 3 is much more appropriate. In particular

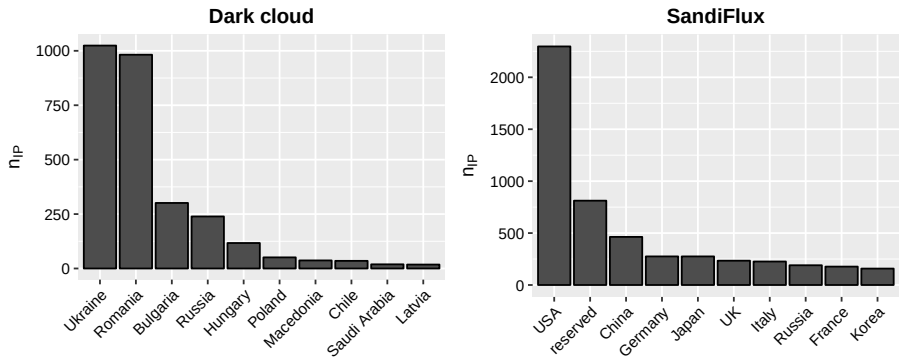


Fig. 4. Histogram of the number of IPs localised in the top 10 countries for the two botnets.

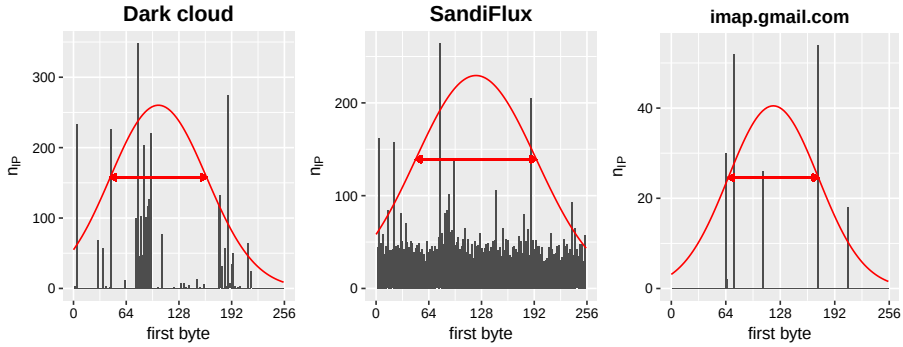


Fig. 5. Histogram of the frequencies of the first byte in the IP-pool associated with the two botnets analysed and with one large CDN (bin-size=2).

$d_{\mathrm{IP}} = 1.2\ 10^{-3}$ for Dark Cloud and $d_{\mathrm{IP}} = 0.69$ for SandiFlux, while the CDN 'imap.gmail.com' has $d_{\mathrm{IP}} = 6.2\ 10^{-6}$.

7 Conclusions

In this paper, we proposed a fast flux detection method based on the passive analysis of the DNS traffic of a corporate network. The analysis is based on *aramis* security monitoring system. The proposed solution has been evaluated over the LAN of a company, with the injection of 47 pcaps associated with 9 different malware campaigns that leverage FFSNs and cover a wide variety of attack scenarios. All the fast flux domains were detected with a very low false positive rate and the comparison of performance indicators with a state-of-the-art work shows a remarkable improvement. An in-depth active analysis of a list

of malicious fast flux domains confirmed the reliability of the metrics used in the proposed algorithm and allowed for the identification of more than 9000 IPs likely to be associated with compromised hosts. These IPs turned out to be part of two notorious botnets, namely Dark Cloud and SandiFlux, to the description of which we therefore contribute.

As a future development, we plan to introduce in the algorithm a metric related to the use of reserved IPs, which we observed to be extensively present in SandiFlux. Another planned development is the inspection of the overlap in terms of IPs among the most suspicious domains, as we saw that many IPs are shared among domains in the same botnet.

References

1. https://www.acs.org.au/content/dam/acs/acs-publications/ACS_Cybersecurity_Guide.pdf
2. https://www.proofpoint.com/us/threat-insight/post/sandiflux-another-fast-flux-infrastructure-used-malware-distribution-emerges
3. https://www.hybrid-analysis.com/
4. https://packettotal.com/
5. https://www.reverse.it/
6. https://virustotal.com/
7. http://www.aramisec.com
8. https://www.malware-traffic-analysis.net/
9. https://tools.ietf.org/html/rfc1035
10. http://www.forbes.com
11. http://www.alexa.com
12. https://dev.maxmind.com/geoip/
13. http://blog.talosintelligence.com/2017/07/threat-roundup-0630-0707.html
14. https://www.torproject.org/docs/tor-manual.html.en
15. http://www.pwncode.club/2017/09/dreambot-targeting-bulgarian-users.html
16. http://blog.talosintelligence.com/2018/03/gozi-isfb-remains-active-in-2018.html
17. https://blog.inthecyber.com/2017/panda-banker-hits-italy-analysis-part-1/
18. https://www.iana.org/assignments/iana-ipv4-special-registry/iana-ipv4-special-registry.xhtml
19. Alieyan, K., Almomani, A., Manasrah, A., Kadhum, M.M.: A survey of botnet detection based on dns. Neural Computing and Applications 28(7), 1541–1558 (2017)
20. Almomani, A.: Fast-flux hunter: a system for filtering online fast-flux botnet. Neural Computing and Applications 29(7), 483–493 (2018)
21. Berger, A., DAlconzo, A., Gansterer, W.N., Pescapé, A.: Mining agile dns traffic using graph analysis for cybercrime detection. Computer Networks 100, 28–44 (2016)
22. Bisio, F., Saeli, S., Lombardo, P., Bernardi, D., Perotti, A., Massa, D.: Real-time behavioral dga detection through machine learning. In: Security Technology (ICCST), 2017 International Carnahan Conference on. pp. 1–6. IEEE (2017)
23. Chahal, P.S., Khurana, S.S.: Tempr: application of stricture dependent intelligent classifier for fast flux domain detection. International Journal of Computer Network and Information Security 8(10), 37 (2016)

24. Crowder, W., Dunker, N.: Dark cloud network facilitates crimeware, https://www.riskanalytics.com/wp-content/uploads/2017/10/Dark_Cloud_Network_Facilitates_Crimeware.pdf
25. Holz, T., Gorecki, C., Rieck, K., Freiling, F.C.: Measuring and detecting fast-flux service networks. In: NDSS (2008)
26. Hsu, C.H., Huang, C.Y., Chen, K.T.: Fast-flux bot detection in real time. In: International Workshop on Recent Advances in Intrusion Detection. pp. 464–483. Springer (2010)
27. Jiang, C.B., Li, J.S.: Exploring global ip-usage patterns in fast-flux service networks. JCP 12(4), 371–379 (2017)
28. Katz, O., Perets, R., Matzliach, G.: Digging deeper – an in-depth analysis of a fast flux network (2017), https://www.akamai.com/us/en/multimedia/documents/white-paper/digging-deeper-in-depth-analysis-of-fast-flux-network.pdf
29. Lin, H.T., Lin, Y.Y., Chiang, J.W.: Genetic-based real-time fast-flux service networks detection. Computer Networks 57(2), 501–513 (2013)
30. Martinez-Bea, S., Castillo-Perez, S., Garcia-Alfaro, J.: Real-time malicious fast-flux detection using dns and bot related features. In: Privacy, Security and Trust (PST), 2013 Eleventh Annual International Conference on. pp. 369–372. IEEE (2013)
31. Metcalf, L.B., Spring, J.: Domain parking: Not as malicious as expected, https://resources.sei.cmu.edu/asset_files/whitepaper/2014_019_001_427477.pdf
32. Nazario, J., Holz, T.: As the net churns: Fast-flux botnet observations. In: Malicious and Unwanted Software, 2008. MALWARE 2008. 3rd International Conference on. pp. 24–31. IEEE (2008)
33. Passerini, E., Paleari, R., Martignoni, L., Bruschi, D.: Fluxor: Detecting and monitoring fast-flux service networks. In: International conference on detection of intrusions and malware, and vulnerability assessment. pp. 186–206. Springer (2008)
34. Perdisci, R., Corona, I., Giacinto, G.: Early detection of malicious flux networks via large-scale passive dns traffic analysis. IEEE Transactions on Dependable and Secure Computing 9(5), 714–726 (2012)
35. Ruohonen, J., Leppänen, V.: Investigating the agility bias in dns graph mining. In: Computer and Information Technology (CIT), 2017 IEEE International Conference on. pp. 253–260. IEEE (2017)
36. Salusky, W., Danford, R.: Know your enemy: Fast-flux service networks. The Honeynet Project pp. 1–24 (2007)
37. Soltanaghaei, E., Kharrazi, M.: Detection of fast-flux botnets through dns traffic analysis. Scientia Iranica. Transaction D, Computer Science & Engineering, Electrical 22(6), 2389 (2015)
38. Stevanovic, M., Pedersen, J.M., DAlconzo, A., Ruehrup, S.: A method for identifying compromised clients based on dns traffic analysis. International Journal of Information Security 16(2), 115–132 (2017)
39. Zhou, S.: A survey on fast-flux attacks. Information Security Journal: A Global Perspective 24(4-6), 79–97 (2015)

ICCST 2017 IEEE

THE 51ST INTERNATIONAL CARNAHAN CONFERENCE ON SECURITY TECHNOLOGY

Madrid, SPAIN

October 23rd – 26th, 2017

MADRID ICCST2017 CARNAHAN

What is ICCST Carnahan?

The **IEEE International Carnahan Conference on Security Technology (ICCST)** is an annual, open forum devoted to promote the exchange of ideas and insights concerning the **security technology industry**. The conference places an emphasis on the **research, development, testing** and **systems engineering of electronic security technology**, including the operational testing, lifecycle and performance. The research publications and scientific papers presented and discussed on this occasion constitute the basis for an **open and informed exchange of knowledge and ideas** concerning the analysis outcomes on both **new and existing technologies as crime countermeasures**.

Our contribution

Our team presented a paper titled "*Real-time behavioral DGA detection through machine learning",* written by our colleagues from the Aramis team.

aramis

In this paper, we report on an effective DGA-detection algorithm based on the network monitoring analysis performed by aramis. The first step of the proposed method is the detection of a bot looking for the C&C and thus querying many automatically generated domains. The second phase consists on the analysis of the resolved DNS requests in the same time interval. The linguistic and semantic features of the collected unresolved and resolved domains are then extracted in order to cluster them and identify the specific bot. Finally, clusters are analyzed in order to reduce false positives. The proposed solution has been evaluated over two networks: (i) an ad-hoc network, where traffic generated with DGA snippets belonging to different DGA attacks was injected and (ii) the LAN of a real company. In the first experiment, we deployed different families of malware employing several DGAs: all the malicious variants were detected by the proposed algorithm. In the real case scenario, the algorithm discovered an infected host in a 15-day-long experimental session, while producing a low false-positive rate during the same period. Thus, the experimental evaluation has confirmed the effectiveness of the proposed approach.

Real-time behavioral DGA detection through machine learning

Federica Bisio, Salvatore Saeli, Pierangelo Lombardo, Davide Bernardi, Alan Perotti, Danilo Massa
aizoOn Technology Consulting
Strada del Lionetto 6
10146, Turin, Italy
Email: [name].[surname]@aizoongroup.com

***Abstract*—During the last number of years, the use of Domain Generation Algorithms (DGAs) has increased with the aim of improving the resiliency of communication between bots and Command and Control (C&C) infrastructure. In this paper, we report on an effective DGA-detection algorithm based on a single network monitoring. The first step of the proposed method is the detection of a bot looking for the C&C and thus querying many automatically generated domains. The second phase consists on the analysis of the resolved DNS requests in the same time interval. The linguistic and semantic features of the collected unresolved and resolved domains are then extracted in order to cluster them and identify the specific bot. Finally, clusters are analyzed in order to reduce false positives. The proposed solution has been evaluated over (1) an ad-hoc network where several known DGAs were injected and (2) the LAN of a company. In the first experiment, we deployed different families of malware employing several DGAs: all the malicious variants were detected by the proposed algorithm. In the real case scenario, the algorithm discovered an infected host in a 15-day-long experimental session, while producing a low false-positive rate during the same period.**

I. Introduction

Cybercrime constitutes one of the most serious threats to the current society, with heavy and sometimes dramatic consequences for many companies, organizations and single individuals [14, 18, 20, 26, 30]. During the last number of years, a key role in cybercrime has been played by botnets: these are networks of compromised computers (popularly referred to as *zombies* or *bots*), which are controlled by a remote attacker (popularly referred to as a *bot herder*) through specific Command and Control (C&C) channels. The strength of the botnet resides in the fact that it can be a highly distributed and highly changeable network, making the tracing and the recovery of all the infected components very difficult, and therefore allowing a secure and stable platform for the implementation of a wide range of malicious and illegal activities such as the spreading of ransomwares, exploit kits, banking trojans, etc. [4, 7, 11, 12, 17, 23, 25, 32].

In botnets, the bot herder and bots can exchange information using different protocols; P2P-based botnets have a more robust C&C structure that is difficult to detect and take down, but they are typically harder to implement and maintain.

In order to combine the simplicity of centralized C&Cs with the robustness of P2P-based structures, many attackers employ HTTP botnets that locate their C&C servers through the dynamic generation of domains using a Domain Generation Algorithm (DGA), also known as domain-flux. With this technique, each bot, using a precalculated seed value known to the bot herder (e.g., the current date), automatically generates hundreds or thousands of pseudo-random domain names that represent candidate C&C domains. The bot sends DNS queries until it connects to the IP address associated to a resolved domain. The key advantage of this strategy is that even if one or more C&C domain names or IP addresses are identified and recovered, the bots will query the next set of automatically generated domains and it will eventually get the IP address of a relocated C&C server. DGA provides therefore a remarkable level of agility and a very resilient communication channel between bots and C&C, making it one of the most used technique in botnet control [6, 7, 8, 12, 15, 24, 33].

For these reasons, DGA detection is of crucial importance in cyber security. A number of different approaches to DGA-detection have been implemented, but DNS-based analysis is one of the most appropriate to obtain quick responses, since it does not need file dumps and it only requires the analysis of a small part of the network traffic (in particular, it can ignore packets' payloads). For this reason, many recent works focused on automatically recognizing DGA within DNS traffic, whenever occurring. Many efforts have been made to use supervised or signature-based approaches [5] but these have obtained limited results in the highly dynamic DGA realm. Therefore, some works have applied unsupervised techniques on DNS traffic data provided by some Internet Service Provider [9, 22, 28, 29] or retrieved by collecting the DNS traffic of a single network [15, 21, 33].

In this paper, we report on an effective DGA-detection algorithm which analyzes the DNS traffic of a single network in *near-real-time*: if we consider a network as the one described in Tab. I, the average execution time of the algorithm is 2 seconds, while the average time between two subsequent runs of the algorithm is 3 minutes and 48 seconds (see Sec. III for more details). The ability to detect an attack in near-real-time is crucial, as it allows for a quick reaction and it is the only way to prevent a potentially severe damage to the company which is using the network under attack [14, 31].

The remainder of the paper is structured as follows. In Sec. II we briefly describe aramis, the monitoring platform which contains the DGA detection method which is the focus

of this paper and which is described thoroughly in Sec. III. Finally, we discuss the experimental results in details in Sec. IV and possible future developments in Sec. V.

II. ARAMIS

The proposed DGA-detection algorithm has been deployed in aramis (Aizoon Research for Advanced Malware Identification System), a network security monitoring platform able to automatically identify a wide range of malware and attacks in near-real-time. aramis software is bundled with dedicated hardware[1], and its structure can be summarized in four phases:

1) Collection: sensors are placed in various nodes of the network. Each sensor gathers the data from its segment of the network, pre-analyzes them in real-time and sends the results to a NoSQL database.
2) Enrichment: inside the NoSQL database, data is enriched with information coming from the aramis Cloud Service, which collects intelligence from various OSINT sources and from internally managed sources.
3) Analysis: two kinds of analyses are executed on the stored data: (i) *advanced cybersec analytics* to spot and highlight specific attacks, among which DGAs, and (ii) a *machine learning engine* which compares the behavior of each node with the usual one.
4) Visualization: the results are presented through cognitive dashboards, which are crucial to highlight anomalies.

The cycle of the four phases restarts after a period Δt which slightly depends on the quantity of analyzed traffic. On the network described in Tab. I $\Delta t = 228 \pm 92$ seconds, which represents the best trade-off between the need of many network data in order to have statistically significant results and the requirement of near-real-time analysis.

III. DGA DETECTION METHOD

The aim of the proposed DGA-detection method is the near-real-time identification of domain-flux attacks via the monitoring of a single network. To this purpose, the method is composed of several steps of analysis:

- Collection of unresolved DNS requests (UNRES): in order to detect a bot trying to connect with the C&C, all the UNRES in a suitable amount of time are collected. The sudden and huge increase of UNRES may in fact indicate the tentative of connection with several untrusted automatically generated domains.
- Filtering and preprocessing of UNRES: all the queries due to user errors (e.g., typos of popular domains) and system misconfigurations are removed.
- Outlier detection: the hosts producing the highest peaks of UNRES are identified.
- Extraction of resolved DNS requests (RES): since a bot stops querying when an existent domain is hit and a successful connection is established, RES near the peaks identified in the previous step are collected.
- Domain features extraction: all the collected RES and UNRES are mapped in a feature space able to embed the linguistic and semantic components (see Sec. III-E).
- Clustering: domains which are similar according to the features extracted in the previous step, are grouped together in order to spot common patterns of the specific bot.
- False positives removal: the level of homogeneity of the clusters is calculated. This allows the distinction between true DGAs (associated with highly homogeneous clusters) from the expected legit unresolved DNS peaks (associated with less homogeneous clusters).

In the following we describe the details of each step.

A. Collection of UNRES

aramis framework operates in near-real-time, therefore all the UNRES are continuously downloaded and analyzed. On the network described in Sec. IV-B, the complete DGA-detection algorithm takes an average time of 2 seconds to complete. Anyway, as discussed in Sec. II, the algorithm is integrated in aramis, which takes an average time Δt of about 3 minutes and 48 seconds to collect all network data and perform the analyses. After this process, aramis produces a detailed analysis of the network risk and restarts a new cycle of analysis.

B. Filtering and preprocessing of UNRES

The following filters are applied to the retrieved UNRES:

- Requests containing invalid or malformed Top Level Domains (TLDs) are removed. Typically, they are due to typos or user errors.
- Overloaded DNS: DNS queries are sometimes overloaded so to provide anti-spam or anti-malware techniques. In order to reduce noise, the overloaded DNS are removed.
- Local and private domains are removed.
- White list domains (i.e., domains that are known to be trusted) are removed.
- Popular domains are removed. More specifically, three popular domains sources are considered: the top 10000 domains in the world provided by Alexa [2], the web URLs of the 500 world biggest companies provided by Forbes [3] and the top 100 domains collected inside the network under analysis. In all these cases, the second and third level domains of an input domain are extracted and compared with the second and third level domains of the list of popular domains; if the Jaro-Winkler distance [27] is below 0.1, the input domain is considered as a misspelling of a popular domain and removed.
- Configuration words: domains containing certain substrings (e.g., words related to network system and structure) are filtered out, because they represent congenital network traffic.
- ARPA domains are filtered out, since they are only used for reverse DNS lookup.

[1]E5-2690 2.9GHz x 2 (2 sockets x 16 cores) 16 x 8GB RAM, 1.1TB HDD

- If a TLD is found in the third or higher levels, it is considered as a misconfiguration of the web browser or of the particular application and hence it is removed.
- If an IP address is found in the third or following levels, it is considered as an internal domain and it is removed.

The filtering phase removes the largest part of the initial UNRES; usually just a 5-10% of the queries are not filtered out and proceed through the other steps of the algorithm.

C. Outlier Detection

In order to recognize burst in the UNRES traffic, time is discretized and the number of UNRES for each machine in each time interval is considered part of a time series, which is described in terms of 6 different statistical methods:

- deviation from the expected distribution calculated via
  - Gaussian estimate
  - kernel density estimate
- arima model [10]
- deviation from the expected behavior calculated on a moving window via
  - mean and standard deviation
  - median and median absolute deviation
  - interquantile range

Each method can be considered as a binary classifier between ordinary points and outliers, and the results of all classifiers are combined with an *ensemble classifier* based on a weighted majority rule[2], which has been shown to perform typically better than any single classifier [13]. The identification of outliers in the distribution of the number of UNRES allows to detect potentially suspicious machines.

D. Extraction of resolved DNS requests

Once the suspicious machines are detected, the extraction of the related RES is performed. In particular, all the RES occurring in a time interval τ around the UNRES peaks are collected. The interval τ is set to 20 seconds; this choice represents a trade-off between the need of a large τ to compensate possible delays in the network data collection and the necessity of a small τ in order to avoid casual associations of RES with a cluster of UNRES.

E. Domain features extraction

The purpose of this phase is the creation of a common feature space for RES and UNRES, able to map into an array of numbers the linguistic peculiarities of the domains under analysis. Pseudo-random domains generated by the same algorithm typically share at least some common linguistic attributes, while legitimate domains are not generated by an algorithm and, hence, should not show similarities in the domain structure. It is known however [15] that some modern DGAs employ English dictionary words with little modifications; for this reason both linguistic and non-linguistic features have been considered [3]. Therefore, the main idea is to extract the most relevant features of both RES and UNRES in order to find common patterns able to characterize a specific C&C connection. In this way, we are able to perform the subsequent clustering phase and group together domains showing a similar pattern, therefore defining the behavior of a particular bot.

The extracted linguistic features are the following:

- Number of levels in the domain
- For the second and third levels: distance of the monograms probability distribution from the one of monograms in the English language
- For the second and third levels: distance of the bigrams probability distribution from the one of bigrams in the English language
- Entropy in characters distribution of the second and third levels
- Number of characters of the second and third levels

TABLE I
NETWORK DESCRIPTION

	Real Network	**Malware Lab**
N. of machines	288	269
N. of clients	209	185
Average N. of connections	136 k / hour	452 k / hour
Average N. of UNRES	791 / hour	14 k / hour
Average N. of RES	59 k / hour	184 k / hour

F. Clustering

Once the domain features are extracted, a k-means clustering [16] is performed on the feature space. The number of clusters N_c is set equal to a fifth of the number of input domains, because this was found as the best trade-off between the need of a large N_c in order to obtain highly homogeneous groups and the need of a small N_c to avoid the separation of domains belonging to the same DGA into many different clusters. Moreover, every cluster has an associated homogeneity value corresponding to the average proximity of the samples of the cluster with the related centroid.

After creating the clusters, malicious clusters have to be recognized; they are identified in the following way:

- Clusters formed by both RES and UNRES and where the number of UNRES is higher than the number of RES;
- Clusters which only contain UNRES.

In both cases, we assign an anomaly indicator A to each malicious cluster proportional to its value of homogeneity. Therefore, A has minimum value $A = 0$ (no anomaly detected) and maximum value $A = 1$ (maximum anomaly detected). The two kinds of clusters contain respectively DGAs which eventually contacted a C&C and DGA attempts which did not find a C&C. Thus, A for the second case is reduced by a corrective factor $\lambda_{\text{fail}} = 0.8$.

[2]The weight used for each method is proportional to the inverse of the mean number of outliers detected by that method: this means that an alarm reported by a method which often presents alarms has a smaller relevance compared to an alarm presented by a usually cautious method.

[3]This choice allowed for the identification of English-dictionary-based DGAs (see Sec. IV-A), e.g., Gozi ISFB, Rovnix, Matsnu, Suppobox.

TABLE II
MALWARE DESCRIPTION AND DETECTION RESULTS

Malware type	Domain layout	Domain length	Malware names	C&C / sinkhole alive	Clusters	Resolved domains	A
Banking Trojan	Alphabetic	Fixed	Fobber [Tinba v3]	Yes	40	2	0.9841
			Ranbyus	Yes	38	5	0.9842
			Tinba [TinyBanker, Zusy]	Yes	29	6	0.9937
		Variable	Qakbot	Yes	71	2	0.9864
			Ramnit	Yes	40	1	0.8885
			Vawtrak [Neverquest, Snifula]	No	82	2	0.9638
	Alphabetic + seed	Fixed	Banjori [MultiBanker 2, BankPatch(er)]	Yes	116	3	0.9955
	Alphanumeric	Fixed	Qadars v3	No	52	1	0.9850
		Variable	Newgoz [Gameover Zeus]	Yes	42	3	0.9926
			Shiotob	No	57	2	0.9615
			ZeusBot	Yes	84	1	0.9731
			Murofet v3 [Licat]	Yes	44	1	0.9859
	Alphanumeric + DDNS	Variable	Corebot	Yes	55	4	0.9847
	Dictionary	Variable	Gozi ISFB[a] [Ursnif, Snifula, Papras]	Yes	30	2	0.9766
			Gozi ISFB[b] [Ursnif, Snifula, Papras]	Yes	51	3	0.9776
			Rovnix	No	174	3	0.9875
Botnet	Alphabetic	Fixed	PushDO [Pandex, Cutwail]	No	63	2	0.9995
	Alphabetic + DDNS	Variable	Kraken v1 [Bobax, Oderoor]	Yes	70	5	0.9834
	Alphabetic	Variable	Necurs	Yes	39	2	0.9664
Exploit Kit	Alphabetic	Variable	Blackhole	No	63	3	0.9924
Ransomware	Alphabetic	Fixed	Cryptolocker	No	24	2	0.9984
			Padcrypt	Yes	84	4	0.9908
		Variable	DirCrypt	No	16	2	0.9784
			Locky v3	Yes	30	3	0.9738
Trojan Horse	Alphabetic	Fixed	Dnschanger [Alureon]	No	57	1	0.9959
			Ramdo	No	131	3	0.9894
			Simda	Yes	34	3	0.9984
			Sisron [TOMB, Trojan.Scar]	Yes	10	1	0.9807
			Srizbi	No	14	1	0.9964
		Variable	Bamital	Yes	21	1	0.9888
			Nymaim	No	48	3	0.9643
	Alphabetic + DDNS	Variable	Vidro	Yes	130	4	0.9866
			Symmi	Yes	49	2	0.9816
	Alphanumeric	Fixed	Chinad	Yes	17	1	0.9861
		Variable	Beped	No	35	2	0.9822
	Dictionary	Variable	Matsnu	No	74	3	0.9897
			Suppobox	Yes	52	1	0.9267
Worm	Alphabetic	Fixed	Tempedreve	No	38	2	0.9877
		Variable	Proslikefan	Yes	98	5	0.9957
			Pykspa [Pykse,Skyper, SkypeBot]	Yes	58	5	0.9835

[a]Employing "luther" dictionary
[b]Employing "nasa" dictionary

G. False positive removal

The anomaly indicator of each cluster is rescaled in order to reduce false positives. The effect of this rescaling is to further decrease low values of A (usually associated with false positives), to highlight large values of A and to enhance the differences in the interval $[0.3, 0.75]$, which has been recognized in the training phase as the overlapping region between the most uncertain false positives and true positives.

IV. EXPERIMENTAL EVALUATION

The DGA-detection algorithm described above was evaluated within two different experimental designs:

- 40 DGA snippets belonging to different malware families were used to inject real DGA network traffic into an ad-hoc network (*malware lab*, see Tab. I). The malware families of the DGA snippets include banker trojans, ransomwares, worms (see Tab. II for a complete list) and cover all the most relevant DGA-attack scenarios.
- The LAN of a real company (described in Tab. I) was observed for a 15-day-long experimental session.

A. Network traffic injection

The first round of experiments consisted in 40 DGA snippets belonging to different malware families used to simulate real DGA traffic inside the *malware lab*, which is described in Tab. I. In order to simulate the successful connection to the C&C, a technique similar to *sinkholing* [19, 8] was used: before the injection of the traffic generated by each snippet, a couple of the domains produced by the snippet were registered in the FakeDns of the *malware lab*. Each registered domain was associated to an IP address of a honeypot running a web

server [4].

Tab. II provides a synthetic description of the malware used in the experiments and the related detection results. For each malware, it contains the following information:

- Malware type
- Domain layout, i.e., elementary components of the generated domains [24]
- Domain length (fixed or variable)
- Specific names of the malware; aliases of the malware names are reported in square brackets
- C&C / sinkhole alive, i.e., a field indicating if at least one domain (in addition to the ones registered as described in Sec. IV-A) was resolved during the experiments
- Number of clusters, i.e., number of groups of similar domains found by the algorithm described in Sec. III-F
- Number of clusters containing resolved DNS requests
- Anomaly indicator A, as described in Sec. III-F

From Tab. II it is possible to notice that the proposed DGA-detection framework successfully detected all the malware variants with a high anomaly indicator. In fact, the value of A averaged on all the samples is 0.9806. Moreover, all the malicious RES have been identified, thus giving the possibility to detect all the active C&Cs, which were reported to the appropriate OSINT repositories.

B. Real company scenario

The second round of experiments was performed through the monitoring of the LAN of a real company, in order to provide a real case test of the proposed solution. In this way, we were able to reliably estimate the false positive rate.

aramis was run for a 15-day-long experimental session. In order to evaluate the performances, we distinguished between RES and UNRES requests. The RES case represents the riskiest situation, since the complete domain-flux attack took place; in this case, therefore, the first concern is the avoidance of false negatives, while some false positives might be tolerated.

On the contrary, the UNRES situation is less risky since it indicates that the potential malware unsuccessfully tried to connect to the C&C. Therefore, in the latter case, a higher false negative rate might be tolerated.

Out of the 285 k unresolved domains of the whole network collected during the observation, the algorithm detected 37 and 1720 domains respectively for the RES and the UNRES case.

We collected results with two different granularities:

- $A > 0$: in this case all the alarms are considered.
- $A > 0.7$: in this case only the alarms with Anomaly Indicator greater than 0.7 (high risk) are considered.

Tab. III reports the obtained results. In particular, it is possible to notice that a real domain-flux attack, including the final contact with the C&C (RES case), has been completely detected. In fact, the alarms associated with this detection were investigated and led to the discovery of the activity of a banking trojan (VawTrak [1]). In this case, as it can be noticed in Tab III, all malicious clusters were detected with $A > 0.7$. Moreover, it is possible to observe that the false positive rate is equal to zero for the RES case, hence allowing to completely distinguish the real attacks from the normal traffic.

[4]Besides the DNS registered in the experiment, other domains were resolved, revealing the presence of active C&Cs or *sinkholes*.

TABLE III
REAL NETWORK RESULTS

UNRES case	$A > 0$		$A > 0.7$	
	DGA	**Not DGA**	**DGA**	**Not DGA**
Detected	1.65 k	70	1.65 k	42
Undetected	0	285 k	0	285 k
RES case	$A > 0$		$A > 0.7$	
	DGA	**Not DGA**	**DGA**	**Not DGA**
Detected	37	0	37	0
Undetected	0	21.3 M	0	21.3 M

Besides, we also analyzed the UNRES case. It is possible to notice that all the 1.65 k malicious UNRES were detected with $A > 0.7$, while the false positive rate is very low: the 0.02% of the 285 k non-malicious UNRES are presented in output by the algorithm, and only the 0.01% has $A > 0.7$.

Therefore, we can conclude that the proposed method is able to detect potentially infected machines in near-real-time and with high anomaly indicators, while limiting the false positives at the same time.

V. CONCLUSIONS

In this paper, we proposed a DGA-detection method based on the analysis of the DNS traffic of a single network; the analysis requires aramis security monitoring system.

The proposed solution has been evaluated over two networks: (i) an ad-hoc network, where traffic generated with DGA snippets belonging to different DGA attacks was injected and (ii) the LAN of a real company. In the first experiment, all the malicious variants were detected, while in the real case scenario the algorithm discovered an infected host. Thus, the experimental evaluation has confirmed the effectiveness of the proposed approach.

As a future development, we plan to refine the clustering algorithm by adding weights to domain levels in the linguistic features extraction phase and by recognizing the layout (alphanumeric or alphabetic). Another aspect that we plan to improve is the filtering phase, with the introduction of a filter for Content Delivery Networks (CDN) and Round Robin DNS (RRDNS), with the development of a CDN/RRDNS identification algorithm. Other developments include the introduction of a specific dictionary for the country where the network is located, the filtering of queries used by some browser (e.g., Chrome and Chromium) to determine if the user is on a network that intercepts and redirects requests for nonexistent hostnames, and the refinement of local domains identification.

REFERENCES

[1] https://www.blueliv.com/downloads/network-insights-into-vawtrak-v2.pdf.

[2] http://www.alexa.com.

[3] http://www.forbes.com.

[4] K. Alieyan, A. ALmomani, A. Manasrah, and M. M. Kadhum. A survey of botnet detection based on dns. *Neural Computing and Applications*, pages 1–18, 2015.

[5] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster. Building a dynamic reputation system for dns. In *USENIX security symposium*, pages 273–290, 2010.

[6] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon. From throw-away traffic to bots: Detecting the rise of dga-based malware. In *USENIX security symposium*, volume 12, 2012.

[7] A. H. R. A. Awadi and B. Belaton. Multi-phase irc botnet and botnet behavior detection model. *arXiv preprint arXiv:1501.03241*, 2015.

[8] T. Barabosch, A. Wichmann, F. Leder, and E. Gerhards-Padilla. Automatic extraction of domain name generation algorithms from current malware. In *Proc. NATO Symposium IST-111 on Information Assurance and Cyber Defense, Koblenz, Germany*, 2012.

[9] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi. Exposure: Finding malicious domains using passive dns analysis. In *Ndss*, 2011.

[10] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[11] D. Dagon, G. Gu, C. P. Lee, and W. Lee. A taxonomy of botnet structures. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pages 325–339. IEEE, 2007.

[12] C. J. Dietrich, C. Rossow, F. C. Freiling, H. Bos, M. Van Steen, and N. Pohlmann. On botnets that use dns for command and control. In *Computer Network Defense (EC2ND), 2011 Seventh European Conference on*, pages 9–16. IEEE, 2011.

[13] T. G. Dietterich et al. Ensemble methods in machine learning. *Multiple classifier systems*, 1857:1–15, 2000.

[14] T. Grance, K. Kent, and B. Kim. Computer security incident handling guide. *NIST Special Publication*, 800:61, 2004.

[15] M. Grill, I. Nikolaev, V. Valeros, and M. Rehak. Detecting dga malware using netflow. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 1304–1309. IEEE, 2015.

[16] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[17] G. Hogben, D. Plohmann, E. Gerhards-Padilla, and F. Leder. Botnets: Detection, measurement, disinfection and defence. *European Network and Information Security Agency*, 2011.

[18] M. Korolov. Cyber security review. *Treasury & Risk*, 2012.

[19] F. Leder, T. Werner, and P. Martini. Proactive botnet countermeasures: an offensive approach. *The Virtual Battlefield: Perspectives on Cyber Warfare*, 3:211–225, 2009.

[20] F. Lemieux. Investigating cyber security threats: Exploring national security and law enforcement perspectives. *2011 Developing Cyber Security Synergy*, page 63, 2011.

[21] M. Mowbray and J. Hagen. Finding domain-generation algorithms by looking at length distribution. In *Software Reliability Engineering Workshops (ISSREW), 2014 IEEE International Symposium on*, pages 395–400. IEEE, 2014.

[22] S. Schiavoni, F. Maggi, L. Cavallaro, and S. Zanero. Phoenix: Dga-based botnet tracking and intelligence. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 192–211. Springer, 2014.

[23] E. Soltanaghaei and M. Kharrazi. Detection of fast-flux botnets through dns traffic analysis. *Scientia Iranica. Transaction D, Computer Science & Engineering, Electrical*, 22(6):2389, 2015.

[24] A. K. Sood and S. Zeadally. A taxonomy of domain-generation algorithms. *IEEE Security & Privacy*, 14(4):46–53, 2016.

[25] M. Stevanovic and J. M. Pedersen. On the use of machine learning for identifying botnet network traffic. *Journal of Cyber Security and Mobility*, 4(3):1–32, 2016.

[26] R. W. Taylor, E. J. Fritsch, and J. Liederbach. *Digital crime and digital terrorism*. Prentice Hall Press, 2014.

[27] W. E. Winkler. String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage. 1990.

[28] S. Yadav, A. K. K. Reddy, A. Reddy, and S. Ranjan. Detecting algorithmically generated malicious domain names. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 48–61. ACM, 2010.

[29] S. Yadav and A. N. Reddy. Winning with dns failures: Strategies for faster botnet detection. *Security and privacy in communication networks*, pages 446–459, 2012.

[30] T. Yadav and R. A. Mallari. Technical aspects of cyber kill chain. *arXiv preprint arXiv:1606.03184*, 2016.

[31] X. Yin, W. Yurcik, Y. Li, K. Lakkaraju, and C. Abad. Visflowconnect: Providing security situational awareness by visualizing network traffic flows. In *Performance, Computing, and Communications, 2004 IEEE International Conference on*, pages 601–607. IEEE, 2004.

[32] H. R. Zeidanloo, M. J. Z. Shooshtari, P. V. Amoli, M. Safari, and M. Zamani. A taxonomy of botnet detection techniques. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, volume 2, pages 158–162. IEEE, 2010.

[33] H. Zhang, M. Gharaibeh, S. Thanasoulas, and C. Papadopoulos. Botdigger: Detecting dga bots in a single network. In *Proceedings of the IEEE International Workshop on Traffic Monitoring and Analaysis*, 2016.

ICPS 2017

INTERNATIONAL CONFERENCE OF PHYSICS STUDENTS

Turin, ITALY

August 7th – 14th, 2017

What is ICPS?

The **International Conference of Physics Students (ICPS)** is the main event organized each year by the **International Association of Physics Students (IAPS)**. The main purpose of the conference is to create an opportunity for physics students coming from all over the world to gather, **discuss science and technology** and practice **presenting their research**.

Keynote speakers include **world-leading scientists and researchers** in multiple areas of physics, while young **students and researchers** have an unparalleled opportunity to present and debate their **research projects and analysis outcomes**. Among the core themes debated on the occasion, we cite quantum physics and **technology, computational physics** and **complex systems**.

Our contribution

Our team presented a scientific poster on "***ARAMIS – AizoOn Research for Advanced Malware Identification System***".

In this poster, we describe the Aramis platform, our solution for network monitoring analysis, entirely designed and developed by aizoOn thanks to the convergent use of advanced Machine Learning algorithms, Threat Intelligence and Advanced Cyber Analytics. Artificial intelligence is globally recognized as a powerful ally in the early detection of advanced cyber threats. The effectiveness of these mathematical tools applied to cyber security depends on the capacity of algorithms to eliminate the "background noise", to read data as an expert analyst would do and to provide the Security Operation Center with monitoring tools and useful information for a fast identification of real threats. Aramis is a scalable network monitoring solution developed to be easily and organically integrated into the ICT Risk Management process.

aramis: Aizoon Research for Advanced Malware Identification System

P. Lombardo, F. Bisio, D. Bernardi
aizoOn Technology Consulting

AUSTRALIA EUROPE USA
TECHNOLOGY CONSULTING

MOTIVATIONS

- Cybercrime is one of the most serious threats to the current society
- The knowledge and implementation of cyber security guidelines is crucial
- Malware and attacks rapidly evolve in time and are very heterogeneous (around 80% of malware found in breach investigations is specific to that organization)

COLLECTION OF NETWORK TRAFFIC

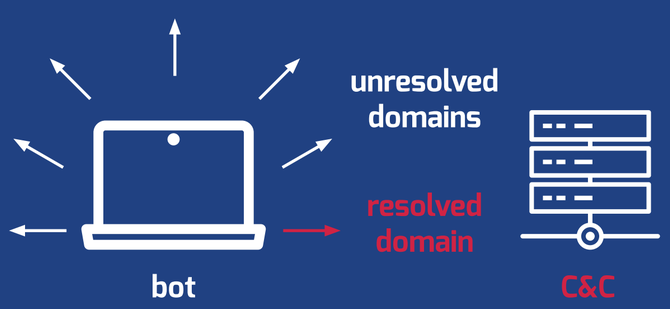
CYBER SECURITY KNOWLEDGE + **DATA MINING TECHNIQUES** → **ADVANCED CYBER ANALYTICS**

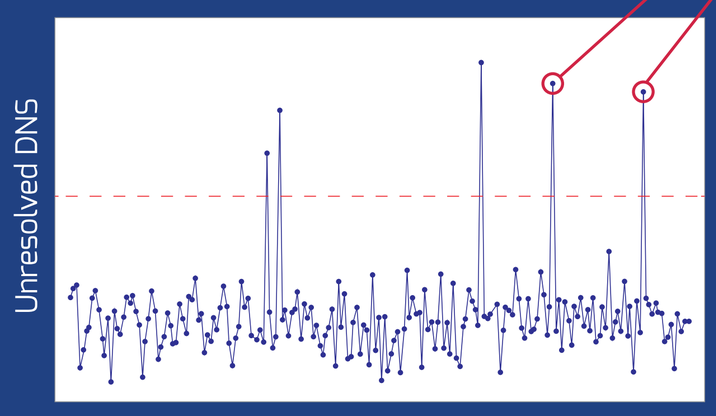
SELF-LEARNED KNOWLEDGE + **MACHINE LEARNING TECHNIQUES** → **MACHINE LEARNING ENGINE**

Each *Advanced Cybersec Analytics* recognizes a specific attack (e.g., **domain generation algorithm[1], drive by download[2], ransomware, IP-flux**) or analyzes a specific aspect of the network traffic (e.g., **network topology, IP geolocation, communication protocol, user agent, scheduled operations, constant data transmission**).

(1) DOMAIN GENERATION ALGORITHM

- Provides a communication channel ***bots*** ⟷ ***command and control (C&C)***
- Each bot generates many pseudo-random domains
- One of them is associated with the C&C and it is resolved

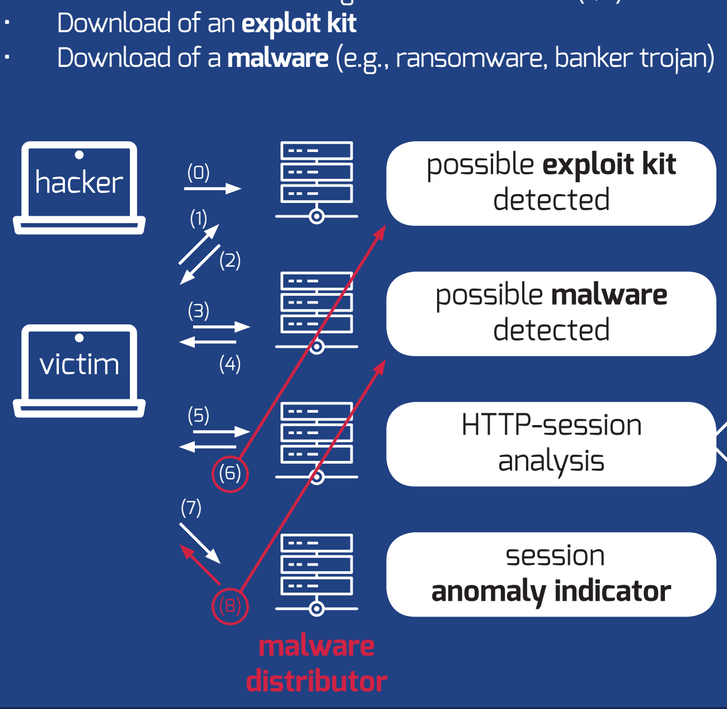




- collection of unresolved domains
- outliers detection
- extraction of resolved domains
- domain features extraction
- clustering (k-means)
- cluster **anomaly indicator**

(2) DRIVE BY DOWNLOAD

- Redirection chain from legitimate web server (1,2) to malware distributor (7,8)
- Download of an **exploit kit**
- Download of a **malware** (e.g., ransomware, banker trojan)

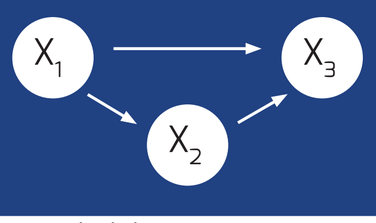


- possible **exploit kit** detected
- possible **malware** detected
- HTTP-session analysis
  - fraction of lecit traffic
  - entropy
- session **anomaly indicator**

The **Machine Learning Engine** analyzes network traffic with two different unsupervised classification algorithms.

BAYESIAN NETWORK

- direct acyclic graph

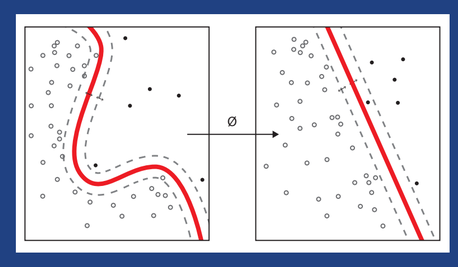


- probabilistic structure $P(X_1, X_2, X_3) = P(X_1)\, P(X_2 | X_1)\, P(X_3 | X_1, X_2)$
- the Bayesian network is built over η variables $X_1, .. X_\eta$

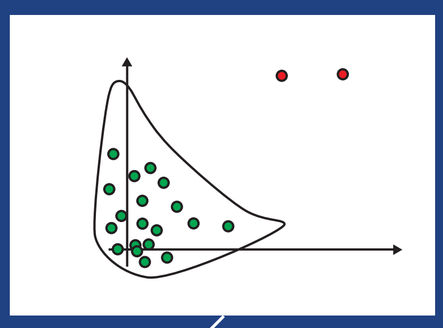
- for each event, for each X_i evaluated
- if $P(X_i | \{X_1, .. \hat{X}_i, .. X_\eta\}) < P_{th}$
- X_i anomalous
- BN **inconsistency indicator** ⟷ # anomalous variables

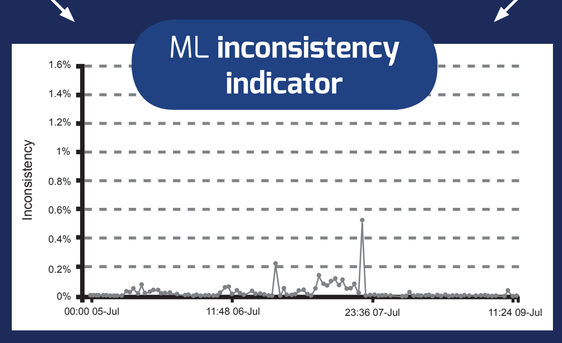
SUPPORT VECTOR MACHINE

- events are points in a multidimensional space
- maximize the *margin*, Gaussian *kernel*



- **one-class SVM**: the training distribution defines the class





Each event is evaluated over:
- a model representing the machine involved
- a model representing a homogeneous class of machines (e.g., clients, servers, etc.)

The variables used for both BN and SVM include communication protocol and service, destination port, duration and volume of the HTTP requests and ws, status code, user agent, etc.

THREAT DETECTION

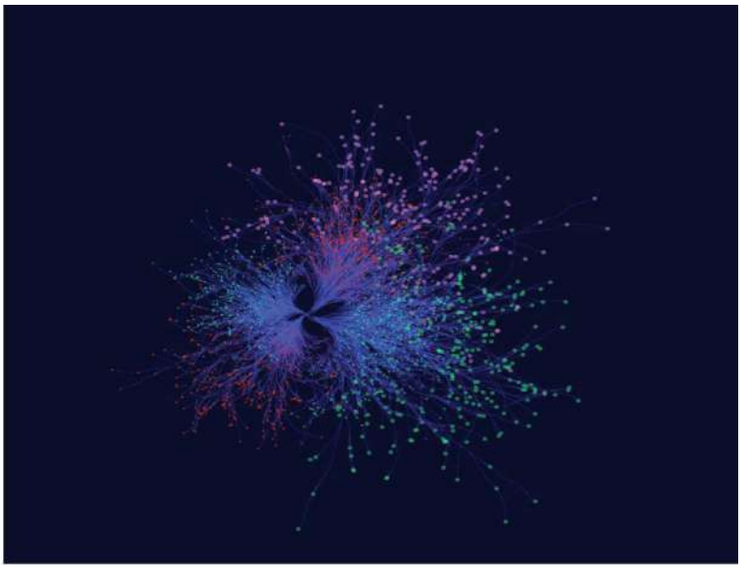
While the *advanced cybersec analytics* automate cybersec experts investigations as much as possible, the *machine learning engine* aims to spot any deviations from the usual behaviors in the network traffic. The combination of these two different approaches allowed for the following **detection results** (found after one month of aramis execution on the network of a medium-size company):

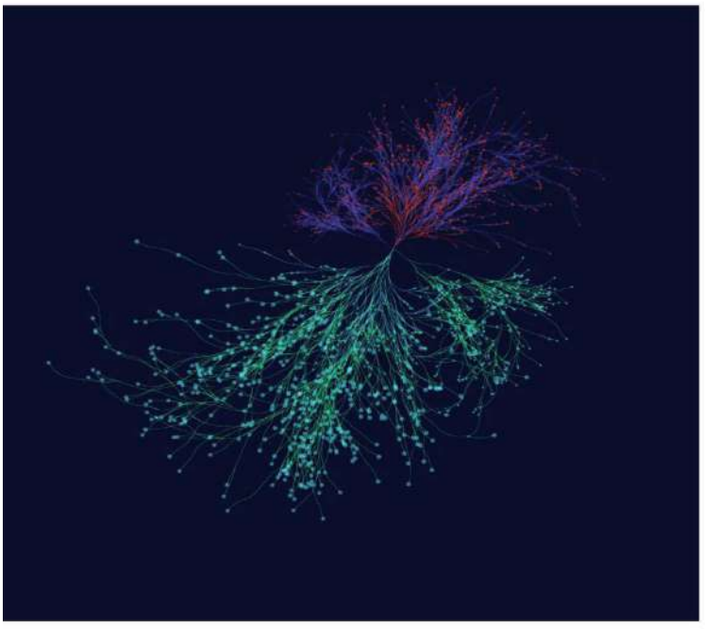
- 1 banking trojan (VawTrak)
- anomalous files exchange ($4 \cdot 10^4$ files/hour) from a client to advertising URLs
- 2 network and resource abuses
- 1 attempt of an Apache PHP remote exploit
- 106 unhautorized TOR connections

Machine learning

Incoherence Index: it represents the **deviation from the «normal» behaviour** of the network.
It is generated by the **Bayesian** algorithms designed by the Data Driven Innovation and Cyber Security teams.

Our Machine Learning engine is based on Bayesian Networks, Support Vector Machine and Bootstrap analysis.





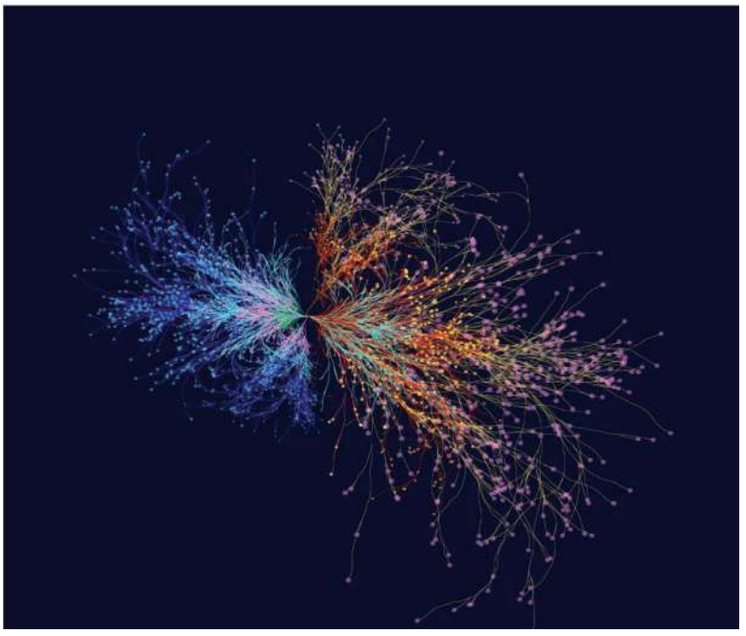
Data mining

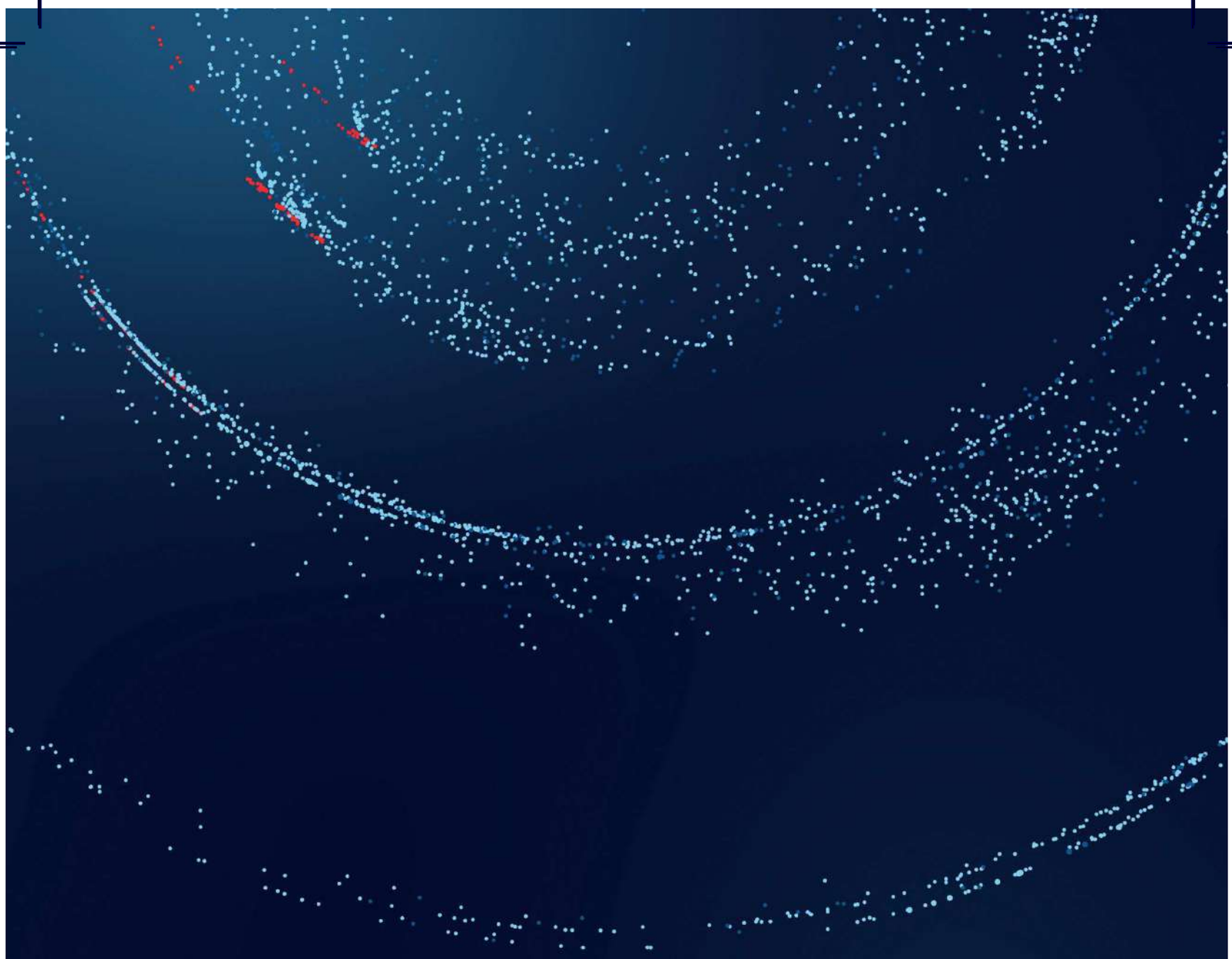
Alarms generated by the **advanced analytics** are developed by the Data Driven Innovation team working closely alongside our security experts.
They are able to do in near real-time more than the analysis that a human network forensic analyst would do. On top of the standard analysis, to check the network traffic connections such as geo-localisation, protocols, quantity of data and so on, our team has developed advanced analytics to spot threats such as Drive by Download, Ransomware, DGA, IP flux and more, with continual study and tuning of the analytics applied to ensure mutations of malicious tool and/or process are incorporated into the engine.

Threat intelligence

Combines the use of publicly available OSINT sources and threat intelligence provided by the **Malware Lab** to identify malicious IPs and DNS, TOR exit nodes and malware inside the traffic analysed by the sensors. We have numerous honeypots around the globe to spot the latest attacking techniques.

The threat intelligence engine gives capabilities such as automatic detection of malware and attacks, scans identification of schemas, correlations and attack patterns.





Get in touch:

APAC

+61 02 8299 7302
aramis@aizoon.com.au

EMEA

+39 06 97605931
aramis@aizoongroup.com

NORTH AMERICA

+1 866 398 6567
aramis@aizoon.us

aramisec.com

aizoon® AUSTRALIA EUROPE USA
TECHNOLOGY CONSULTING