International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 5, No.1 (Jan-2016)

http://dx.doi.org/10.12785/ijcds/050104

A Network Traffic Representation Model for Detecting Application Layer Attacks

Enrico Cambiaso[1,2], Gianluca Papaleo[1], Giovanni Chiola[2], and Maurizio Aiello[1]

[1] CNR-IEIIT, National Research Council, via De Marini 6, 16149 Genoa, Italy
[2]DIBRIS, Università degli Studi di Genova, via Dodecaneso 35, 16146 Genoa, Italy

Received 1 Sep. 2015, Revised 23 Oct. 2015, Accepted 15 Nov. 2015, Published 1 Jan. 2016

Abstract: Intrusion Detection Systems (IDS) play an important role in network security, protecting systems and infrastructures from malicious attacks. With the emerging of novel threats and offensive mechanisms, IDS require updates in order to efficiently detect new menaces. In this paper we propose an anomaly-based detection model designed for particular application protocols, exploited by emerging menaces known as Slow Denial of Service (DoS) Attacks. We define parameters characterizing network traffic and we describe in detail how to extrapolate them from a network traffic capture. We motivate the need of packet inspection in certain contexts in order to retrieve correct data. We analyze and describe how the proposed model behaves on two real scenarios involving legitimate and malicious activities, respectively. Thanks to our model, a detection framework for attacks working at the application layer of the communication protocol stack is provided, allowing and facilitating the execution of detection algorithms. Indeed, though the adoption of such framework, the design of efficient detection systems is simplified and designers work is reduced, allowing them a faster deploy of efficient detection algorithms. The aim of this paper is to provide an effective framework for application DoS attacks detection.

Keywords: intrusion detection, anomaly detection, detection model, framework, lbr dos, slow dos attack

1. INTRODUCTION

In the last century, communication has evolved and Internet became the most relevant communication medium. As today Internet-connected computer systems play a vital role in modern society, they are often subject to intrusions and attacks. Therefore, Internet has to be kept a safe place, providing an appropriate security layer to its users. Intrusion detection techniques are executed to identify malicious activities targeting a specific network or host.

Intrusion Detection Systems (IDS) can be categorized into *anomaly detection* and *misuse detection*: while anomaly detection systems, such as IDES [1], flag as anomalous each activity that significantly deviates from normal usage profile, misuse detection systems, such as IDIOT [2] or STAT [3], profile well-known menaces extrapolating attack signatures characterizing an intrusion.

Currently, building an effective IDS is no easy task. An anomaly-based approach may use intuition and experience to identify statistical measures [4], while a misuse approach first analyzes and categorizes attacks and vulnerabilities, thus defining specific rules and patterns to identify a running threat. Once the signature of a particular menace is obtained, a potential execution of the same attack could successfully be detected. Nevertheless, since such signature based approach cannot detect novel attacks, it should not be considered a complete solution.

In this paper, we propose a novel representation model for application-layer attacks, focusing on denial of service (DoS) threats. We will particularly refer to Slow DoS Attacks (SDAs) [5], [6], also known as application DoS or low-rate DoS attacks, which represent an emerging category of denial of service attacks. Examples of such threats are the Shrew [7], Slowloris [8], Apache Range Headers [9], the recent HTTP.sys PoC (MS15-034) [10], Slow Read [11], and SlowDroid [12] attacks. It is important to highlight that, although we will contextualize the model for analyzing SDAs to HTTP servers, it is designed to potentially detect application layer attacks in general. Indeed, the proposed system extracts accurately selected features able to characterize application layer threats. This extraction could generally be adopted for any application-layer protocol working over the TCP transport protocol and based on the so called *request-response* (or *request-reply*) [13] communication mechanism (i.e., HTTP, SSH, SMTP, etc.). The proposed system is

E-mail address: enrico.cambiaso@ieiit.cnr.it, gianluca.papaleo@ieiit.cnr.it, chiolag@acm.org, maurizio.aiello@ieiit.cnr.it

intended to be a fundamental component of an IDS. It is indeed designed to be extended and integrated with a wide range of detection algorithms, applying specific approaches to the data extracted through our system.

The rest of the paper is organized as follows: Sect. 2 presents the related work on the topic. Sect. 3 justifies the need of a detection framework for application-layer threats, and Sect. 4 describes in detail our proposed model. Sect. 5 reports the tests we have executed and describe the obtained results, while Sect. 6 reports an example of application of our detection framework. Finally, Sect. 7 concludes the paper.

2. Related Work

The idea of an intrusion detection framework is not new. In [14], a data mining framework for adaptively building intrusion detection models is provided. Other works are instead focused on designing a Collaborative Intrusion Detection System (CIDS) framework. In this context, [15] represents a first attempt to categorize CIDS systems. While systems such as DIDS [16], DShield [15], or NSTAT [17] present centralized CIDS systems, in [18] [19] a hierarchical approach is used. Instead, [20] and [21] propose CIDS systems based on a fully distributed approach.

Other works are instead focused on designing a framework for detecting attacks targeting particular networks. Special interest is given to wireless systems: [22] introduces an IDS framework for cluster-based wireless sensor networks, while [23] aims at protecting wireless sensor networks from routing attacks. [24] proposes a distributed IDS framework for MANETs based on cooperation among nodes, while [25] introduces a layered intrusion detection framework for ad-hoc networks. [26] integrates a layered intrusion detection framework with neural network algorithms to design an IDS system. [27] combines data mining classification with clustering fundamentals to design a hybrid intrusion detection framework.

In order to design an intrusion detection framework for anomaly detection, a proper identification of important features is needed. Although [28] tries to identify those features, the choice of important parameter is still an open challenge in the field. In [29] a statistical intrusion detection framework known as SID is proposed, identifying a set of flow based parameters (duration, protocol, service, source port number, and source byte) used during the detection process. The model proposed in this paper identifies traffic features working at the application layer extracting parameters from connections established with the victim. In this context, [30] focuses on data at the application layer, analyzing information stored in daemons log files. Although this approach may reveal anomalies working at application layer, it is not designed to efficiently protect from slow DoS attacks. Indeed, for example, under a slow DoS attack (i.e. slowloris), server logs may not reveal anomalies, since the attacker's behavior is apparently legitimate. In addition, since logs records may be stored only once (malicious) connections are closed (i.e. identification of a malformed request), such approaches fail in on-line timely protection. Conversely, the proposed method can be applied for on-line detection of running threats.

In [31] it is presented instead an application layer attacks detection system based on re-distributing clients requests on multiple virtual servers. This load-balancing approach, among with the introduction of a network proxy or a network accelerator, is known to counter some specific application threats [32]. Similar protection approaches, such as mod-security or reqtimeout modules of Apache, are able to mitigate specific slow DoS threats. Nevertheless, as demonstrated in [32], these techniques are not designed to protect from distributed threats. Instead, the proposed system implicitly considers distributed attack perpetrated by many coordinated nodes/bots. The approach followed by [33], [34] is instead based on dynamically changing the port number of the server through a *random port hopping* approach. We believe that such approach should be considered invasive for the server, while we prefer adopting a technique which can be deployed even on a network node such as a firewall. In [35] a mechanism for protecting web servers from application DoS attacks is proposed, focusing on a whitelist-based admission control and busy period-based attack flow detection. [36] proposes instead a detection system to counter the LoRDAS attack, exploring different approaches to defend against this threat and suggesting approaches based on Random Service Time (RST), Random Time Queue Blocking (RTQB), Improved Random Time Queue Blocking (IRTQB) and Random Answer Instant (RAI). [37] introduces a low-rate DoS attacks intrusion detection system making use of signal processing based models applied on spectral energy distribution probability. The system is based on analyzing incoming network traffic on the server. Nevertheless, since some low-rate threats induce the server to an anomalous behavior [5], the system is not able to properly identify such threats. The approach adopted in [37] is based on analyzing packets timing arrival. [38] proposes a detection system specific for SQL injection attacks, based on the analysis of repetition of characters. Similarly to [37], the proposed system is based on the extraction of various data relative to the captured packets.

The anomaly detection model we introduce in this paper is based on the extraction of specific metrics able to characterize (distributed or not) application layer menaces. The metrics definition and extraction phase is a fundamental and crucial activity researchers have to accomplish in order to design an efficient intrusion detection system. Focusing on the extraction of relevant data for application DoS attacks detection, the proposed system staves researchers off the need of such activity, opening to a wide range of implementations, integrating algorithms belonging to different research areas. In virtue of this, the proposed system should be considered a fundamental tool for the cybersecurity research field.

3. THE NEED FOR A DETECTION MODEL

Our goal is to design a representation model based on parameters able to characterize network anomalies related to an exploitation of the application layer. Since application layer attacks such as slow DoS threats are becoming very common, an efficient detection system able to identify them is needed. Although the menaces work at the application layer of the communication protocol stack, the identified extrapolated parameters refer to lower layers as well. For instance, at lower layers it is possible to retrieve information related to higher layers, such as the application layer payload size of the packet, analyzed at the transport layer.

Analyzing the available detection and mitigation methodologies for network attacks, a protection system may involve hardware or software components. While applying a hardware protection (such as a load balancer, a network proxy, or a hardware accelerator [32]) could successfully mitigate some categories of attacks, it can be considered a workaround rather than a good solution. Indeed, since such hardware appliances have not been designed for this purpose, their resources would be allocated for this additional activity. Moreover, such approaches usually do not provide a built-in functionality aimed at detecting a working attack on the network.

Instead, if we consider the HTTP protocol, which is particularly exploited by network attacks such as SDAs, current software protections systems are organized as software modules. Although there are several different modules available, their functioning is based on two basic principles: (i) limit the maximum number of simultaneous connections coming from a particular client, and (ii) apply specific server-side timeouts [32], [39]. Even if such solutions represent some sort of mitigation techniques, [32] shows the inefficacy of the available modules in protecting from particular attacks such as, e.g., distributed attacks.

If we analyze the design process for building an IDS, three core activities are involved:

1. first of all, a behavior is observed, usually on the server host or on the server's network path. Based on this observation, a network traffic *representation* phase is accomplished to select specific parameters to extrapolate, in order to identify potential network anomalies
2. subsequently, an *analysis* algorithm is applied to the extrapolated data, by using approaches and metrics deriving from different research fields such as machine learning, neural network, statistics, game theory, etc.
3. finally, a *characterization* phase is accomplished, by defining a proper threshold distinguishing a legitimate situation from an anomalous one. Also in this case, different research areas may be involved.

Our proposed model belongs to the first category of activities and it provides a representation framework specific for DoS attacks working at the application layer. Since we believe that a component-based implementation of an IDS is fundamental, our proposed model not only provides important parameters able to identify application DoS attacks, but it also simplifies researchers' work, allowing them to use an already defined representation system.

4. THE PROPOSED MODEL

Our model is specific to request-response protocols acting at the application layer and making use of the TCP transport protocol. At this layer, the protocol is based on messages exchange between two entities, commonly known as client and server. Moreover, we assume that for each message sent by the client there is always one and only one reply message sent by the server. Known examples of request-reply protocols are HTTP, FTP, or SSH.

A. *Connections on Request-Response Protocols*

After the connection between client and server has been established, the client sends a request to the server. The request is interpreted by the server in order to generate a response to send back to the client. After the first request-response exchange, two possible events could characterize the connection: (i) the connection is closed or (ii) the connection is kept alive (*persistent connection* [40]), in order to reduce the connection overhead for any additional request-response between the same client/server pair.

B. *The Extracted Parameters*

Since each connection potentially is persistent, a first extrapolated parameter is the N_{req} parameter, reporting for each connection the number of requests included in each connection stream.

We define as *connection slot* the portion of a connection which refers to the time passing between the start of a request and the end of the relative response on the same stream. According to Fig. 1, let us define $t_{start\_connection}$ the connection start identified by the 3-way-handshake completion, t_{start_req} the starting time of a request, t_{end_req} the ending time of a request, t_{start_resp} the starting time of a response, and t_{end_resp} the ending time of a response. From these values we can extrapolate data relatively to the time passed before sending the first request (Δ_{start}), the duration of a request (Δ_{req}), the duration of a response (Δ_{resp}), and the time passed between the end of a request and the start of the relative response (Δ_{delay}). While the Δ_{start} parameter is associated with a single connection, other parameters are related to each connection slot, which also includes the time passed between the end of the response and the start of the next request of the same stream (Δ_{next}). In addition, since connections slots are part of a connection, each connection is uniquely identified by the C_{id} value (in our case, a sequential integer value), and each connection slot is associated with the C_{id} connection it belongs, plus the slot index S_i on the connection.

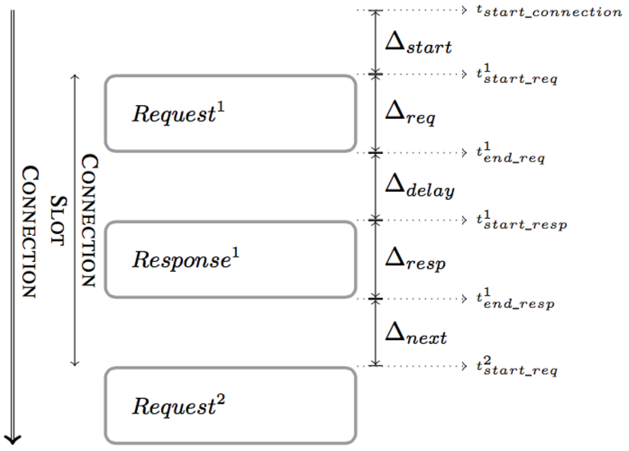

Figure 1. TCP Connection Stream for a Request-Response Protocol Based Connection

Fixing a connection stream, let us define $i = S_i$ the connection slot index on the stream. According to Fig. 1 (where $i = 1$), which depicts a scheme of the parameters, we define:

$$\Delta_{start} = t^1_{start\_req} - t^1_{start\_connection} \tag{1}$$

$$\Delta_{req} = t^i_{end\_req} - t^i_{start\_req} \tag{2}$$

$$\Delta_{delay} = t^i_{start\_resp} - t^i_{end\_req} \tag{3}$$

$$\Delta_{resp} = t^i_{end\_resp} - t^i_{start\_resp} \tag{4}$$

$$\Delta_{next} = t^{i+1}_{start\_req} - t^i_{end\_resp} \tag{5}$$

Moreover, relatively to a single connection slot, the model also provides the following parameters:

- s_{req} to identify the request size, in bytes
- p_{req} to identify the amount of TCP packets that compose a request
- s_{resp} to identify the response size, in bytes
- p_{resp} to identify the amount of TCP packets that compose a response

In particular, let us note that p_{req} and p_{resp} values depend on the data-link layer protocol adopted.

From these parameters we build two tables. Selected parameters are indeed grouped in two categories: from one side, we analyze each connection, extrapolating the triple (i) C_{id}, (ii) N_{req}, (iii) Δ_{start}. These data are part of the *Connections Table*. From the other side, we analyze each slot composing a connection, extrapolating the following data: (i) C_{id}, (ii) S_i, (iii) Δ_{req}, (iv) Δ_{delay}, (v) Δ_{resp}, (vi) Δ_{next}, (vii) s_{req}, (viii) s_{resp}, (ix) p_{req}, (x) p_{resp}. These data are part of the *Connections Slots Table*.

By choosing these parameters, we are able to extrapolate behavioral features for application layer attacks. For instance, it is known that Slow DoS Attacks like Slowloris [8] split HTTP requests by sending request packets delayed during the time, thus being typically characterized by high Δ_{req} values. Instead, the Apache Range Headers [9] attack makes the Δ_{delay} parameter assumes high values, since requests sent to the server need particularly intensive calculations to produce an appropriate response. Similarly, Slow Read attack [11] simulates a tiny reception buffer to slow down the responses of the server, thus being characterized by high Δ_{resp} and p_{resp} values. Relatively to the Δ_{next} parameter, it is instead exploited by the Slow Next attack [41].

Our model provides us the ability to retrieve composed parameters. For instance, the amount of bytes per second sent during a request/response may be defined as reported in Eq. 6.

$$r_{s_{req}} = \frac{s_{req}}{\Delta_{req}} \qquad r_{s_{resp}} = \frac{s_{resp}}{\Delta_{resp}} \tag{6}$$

Similarly, the ratios representing the amount of packets per second sent during a request/response are reported in Eq. 7.

$$r_{p_{req}} = \frac{p_{req}}{\Delta_{req}} \qquad r_{p_{req}} = \frac{p_{req}}{\Delta_{req}} \tag{7}$$

It is also possible to obtain the average size per packet, as reported in Eq. 8.

$$\mu_{s_{req}} = \frac{s_{req}}{p_{req}} \qquad r_{s_{resp}} = \frac{s_{reps}}{\Delta_{resp}} \tag{8}$$

C. Model Assumptions

In order to properly define a detection model, we have to define the model behavior at limit cases. In particular, we make the following assumptions:

- a connection which does not start with a request is ignored until a request is found on the same connection; this may happen when traffic capture operation begins after a full request has been sent and the relative response is captured;

- due to the nature of elements like network, communication medium, or response production times, traffic measurements always provide:

$$\Delta_{delay} > 0 \qquad\qquad \Delta_{next} \neq 0 \tag{9}$$

D. *Messages Overlapping on the Same Connection Stream*

If we focus on the Δ_{next} parameter for a single TCP connection stream, in case $\Delta_{next} < 0$ a connections slots overlapping occurs. Moreover, in this case connection persistence is adopted and connections may include more than a single request. For example, in Fig. 2, the request next to the current one on the same connection stream is (even partially) received before the full response to the current request is sent.

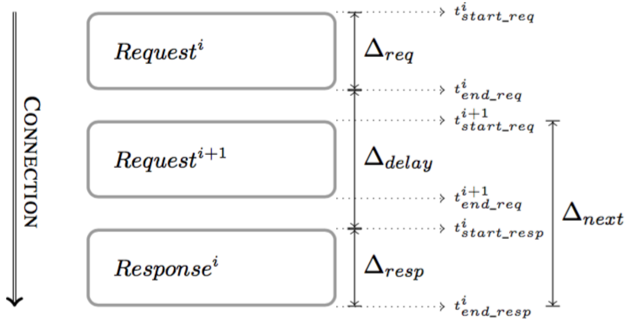

Figure 2. TCP Connection Stream in case of $\Delta_{next} < 0$

Although in Fig. 2 no overlapping is shown between $Request^{i+1}$ and $Response^i$, since $t^{i+1}_{end\_req} < t^i_{start\_resp}$, overlappings may occur. In particular, relatively to a single TCP connection, an overlapping occurs when Eq. 10 is satisfied.

$$\exists j \in N^+ \exists k \in \{j-1, j+1\} | \left(t^k_{start\_req} < t^j_{start\_resp} < t^k_{end\_req}\right) \vee \left(t^k_{start\_req} < t^j_{end\_resp} < t^k_{end\_req}\right) \tag{10}$$

For instance, this kind of overlapping may occur in the HTTP protocol: considering a single TCP connection stream, a client requests to the server a particular resource such as a web page. After the page content is received and parsed by the client, a set of resources bounded to the same hosting server (i.e. pictures, scripts, or stylesheets) may be found in the page. In order to correctly show the web page such additional resources have to be obtained, hence additional requests have to be sent to the server. In case a persistent connection (HTTP 1.1) is used, a series of subsequent requests is usually sent to the server through the already established TCP connection[1]. In this case, the requests sent from the client and next to the first one may be overlapped to the receiving of the responses to the previous requests.

Our model is based on the concept that each connection is composed by a sequence of requests and responses. This fact is not always true; as described above, in case of messages overlapping at the application layer some connections may adopt a full-duplex communication thus resulting in a simultaneous communication between client and server. The model has therefore to adapt itself to correctly identify the start/end of a request/response.

In this context, although a first implementation may identify changes in the packets direction for a common connection stream, such solution may generate inaccurate data. Indeed, although in this case packet inspection at the application level is not needed, if message overlapping occurs improper data would be generated, due to the (possible) frequent change of direction relative to two different and overlapping messages on the same channel. Therefore, in order to carefully extrapolate connection slots data an external protocol-dependent module may be needed.

E. *HTTP Model Implementation*

After defining the model, we implemented a software component to extrapolate the selected representation features. Our intent is to analyze traffic data, such as a PCAP packet capture file, representing the rough network packets on a network/host relatively to a time interval. The analyzed rough data are only relative to the packets directed to the application layer. From such packets it is possible to rebuild and extrapolate a list of (captured) connection streams. This choice allows us to easily retrieve needed capture files by sniffing the network on the server needing protection. Nevertheless, some issues are related to such approach: due to network traffic dump limits, a request (or similarly a response) composed by a single packet leads to:

$$t_{end\_req} = t_{start\_req} \Longrightarrow \Delta_{req} = 0 \tag{11}$$

since capture files associate a packet to a specific reception time. For instance, this fact may occur in case of a single packet including the entire request payload. Although this issue may be considered an important limitation, a more accurate retrieval is not relevant in this context, since our model focuses on attacks targeting the application layer, while in case of a single packet composing the request (or similarly for a response), we expect potentially long Δ_{req} values for attacks targeting lower layers. In addition, a more accurate data retrieval would operate on the protected server (for instance by operating at the kernel level, by intercepting sent and received messages), thus excluding a central node analyzing an entire subnetwork.

[1] Actually, some browsers use multiple connections in order to speed up displaying

Another important consideration is relative to packets payload. In particular, the inspection of the messages directed to the application layer is needed for protocols allowing message overlapping (see Sect. 4.D). Indeed, in this case an analysis of the payload is needed in order to identify the starting/ending times of a request/response on a mixed stream. For instance, in case of the HTTP protocol the end of requests is identified by analyzing packets payloads and looking for the `\r\n\r\n` string. Instead, the `Content-Length` value sent by the server in the response header is needed to identify the response end. Conversely, in case messages overlapping is not supported by the protocol, inspection is not needed, and it is possible to identify requests/responses times by analyzing the direction flow of the packets. Although the packet inspection requirement is a limitation of the proposed approach, gaining access to the server needing protection should not be a problem, hence messages decryption should be possible (i.e. making use of private encryption keys).

F. Model Architecture

Figure 3 shows that the first step of the model extrapolates the observed data. For instance, in our implementation these data are represented as rough network traffic (a PCAP packet capture file). For retrieval operations, traffic has been filtered on packets directed to the application layers, except for the retrieval of the Δ_{start} parameter, requiring the 3-way-handshake packets. Then the data extrapolation phase is accomplished by applying payload inspection, if needed, in order to manage messages/connection slots overlapping possibly occurring on the same connection stream. Two separate tables are generated: the Connections Table, including a connection identifier, the number of requests found in the stream, and the associated Δ_{start} value, for each connection establishment captured; and the Connections Slots Table, where each row refers to single connection slot on the stream and includes the parameters defined in Sect. 4.B. These tables represent the output of the proposed model and are taken in input from a detection algorithm which makes use of statistical approaches to identify traffic anomalies.

Moreover, due to the definition of the N_{req} parameter we always have a number of requests/connection slots greater than or equal to the number of captured connections. This is because every connection always includes at least one connection slot. Therefore, since each connection slot is associated with a single record of the Connections Slots Table, the number of records composing this table is equal to the sum of the values composing the Connections Table, whose values are strictly greater than 0.

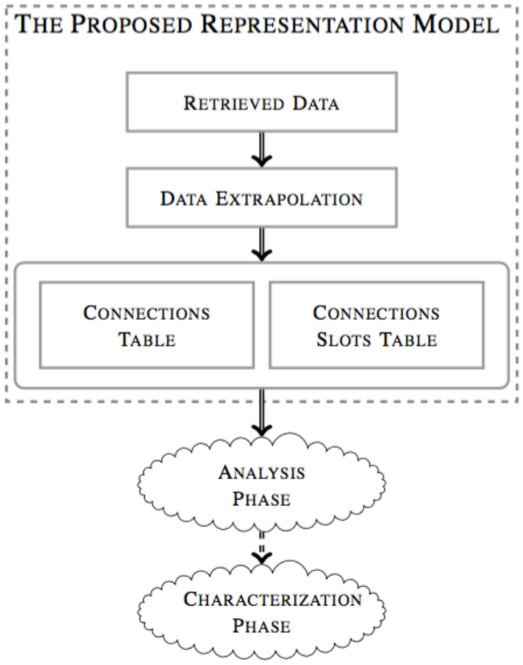

Figure 3. The Proposed Model Architecture

5. TESTS AND RESULTS

We have used our proposed model in a real environment (considering the HTTP protocol), executing tests with the purpose of showing the model's potentialities. Tests involved the capture of network traffic on an Apache HTTP server without any specific protection module running on it.

Two different scenarios have been analyzed: in the former all the traffic directed to the server is legitimate; in the latter only malicious packets are received by the server, generated by a Slowloris [8] attack. Since this attack works by establishing a large amount of connections with the victim and making pending and potentially endless requests, we expect in this case to obtain different values, especially for the Δ_{req} parameter.

In order to show the whole obtained tables generated by our proposed model, we have limited the number of requests accomplished during the tests. In particular, in the legitimate scenario a web page has been requested for $N_{legitimate} = 10$ times during the capture, disabling client-side caching and using Google Chrome web browser. The requested page contains two additional resources located on the same server: a CSS stylesheet and a JPEG picture. We captured traffic relative the $N_{legitimate}$ page openings.

Instead, in the Slowloris attack scenario no specific resource is actually requested to the server: indeed, requests are never parsed by the server since the string identifying the end of the request is never sent by the malicious client. In this case we have limited the connections number to $N_{slowloris} = N_{legitimate} = 10$ and we have applied a Wait Timeout of $T_{WT} = 60$ seconds [6]. Network traffic has been captured for $T = 600$ seconds. Although the limits used during our tests may represent not harmful situations, our intent is to show the results of the proposed method instead of comparing a legitimate situation with several clients to a fully working attack situation.

A. *Legitimate Traffic Analysis*

In this case we expect to capture $N_{legitimate}$ persistent connections in total, thus a Connections Table composed by 10 records. In practice, as reported in Fig. 4, we noticed that more than $N_{legitimate}$ connections are established: we observed a total of 12 established connections and not all of them exactly include 3 requests/connection slots, which represents the expected value, since pages are composed by the HTML content, the CSS stylesheet, and the JPEG picture. In particular, deeply analyzing the network traffic we discovered that connection with $C_{id} = 7$ is relative to the web page favicon, automatically requested by Google Chrome browser, even if not specified in the HTML content.

Moreover, although they are not displayed in sequence (since order is not relevant in our context), connections with $C_{id} = 2$ and $C_{id} = 10$ refer to the same web page show. In particular, the latter is relative to HTML web page and CSS stylesheet requests, while the former is relative to the JPEG picture request. This behavior is due to the web browser used during the tests. In particular, the browser may open more than one connection for showing the same web page for performance optimization, by separating typically large-sized resources (such as images, video, or audio ones) from small sized ones (such as page contents, stylesheets, or scripts).

Figure 4 also reports results relatively to the Connection Slots Table built from the captured traffic. In this case the Connections Slots Table is composed of 31 connection slots. In particular, three requests are accomplished at any page showing, thus making in total $3 \cdot N_{legitimate} = 30$ requests (plus the favicon request automatically issued by the browser).

C_{id}	N_{req}	Δ_{start}
1	3	0.00094
2	1	0.00093
3	3	0.00075
4	3	0.00093
5	3	0.00098
6	3	0.00092
7	1	0.00116
8	3	0.00090
9	3	0.00254
10	2	0.00097
11	3	0.00076
12	3	0.00092

Connections Table

C_{id}	S_i	Δ_{req}	Δ_{delay}	Δ_{resp}	Δ_{next}	s_{req}	p_{req}	s_{resp}	p_{resp}
1	1	0.00000	0.00075	0.00000	0.02675	400	1	816	1
1	2	0.00000	0.00052	0.00000	0.00337	419	1	432	1
1	3	0.00000	0.00034	0.00816	–	418	1	52169	38
2	1	0.00000	0.00045	0.00542	–	418	1	52170	38
3	1	0.00000	0.00091	0.00000	0.03586	400	1	816	1
3	2	0.00000	0.00054	0.00000	0.00298	419	1	432	1
3	3	0.00000	0.00034	0.00570	–	418	1	52169	38
4	1	0.00000	0.00076	0.00000	0.02707	400	1	816	1
4	2	0.00000	0.00053	0.00000	0.00237	419	1	432	1
4	3	0.00000	0.00033	0.00826	–	418	1	52169	38
5	1	0.00000	0.00074	0.00000	0.06189	400	1	816	1
5	2	0.00000	0.00056	0.00000	0.00152	419	1	432	1
5	3	0.00000	0.00034	0.00559	–	418	1	52169	38
6	1	0.00000	0.00073	0.00000	0.05575	400	1	816	1
6	2	0.00000	0.00055	0.00000	0.00175	419	1	432	1
6	3	0.00000	0.00034	0.00565	–	418	1	52169	38
7	1	0.00000	0.00067	0.00000	–	316	1	506	1
8	1	0.00000	0.00090	0.00000	0.02832	400	1	816	1
8	2	0.00000	0.00054	0.00000	0.00268	419	1	432	1
8	3	0.00000	0.00034	0.00709	–	418	1	52169	38
9	1	0.00000	0.00074	0.00000	0.04879	400	1	816	1
9	2	0.00000	0.00056	0.00000	0.00275	419	1	432	1
9	3	0.00000	0.00035	0.00681	–	418	1	52169	38
10	1	0.00000	0.00074	0.00000	1.47218	400	1	816	1
10	2	0.00000	0.00055	0.00000	–	419	1	432	1
11	1	0.00000	0.00072	0.00000	0.02797	400	1	816	1
11	2	0.00000	0.00054	0.00000	0.00355	419	1	432	1
11	3	0.00000	0.00034	0.00599	–	418	1	52169	38
12	1	0.00000	0.00077	0.00000	0.03167	400	1	816	1
12	2	0.00000	0.00054	0.00000	0.00217	419	1	432	1
12	3	0.00000	0.00042	0.00671	–	418	1	52169	38

Connections Slots Table

Figure 4. Results Obtained for Legitimate Traffic

Some of the Δ_{next} values reported in the table assume null values. Indeed, this happens when no requests next to the current one are observed. Moreover, from the Connections Slots Table we can observe that each request is composed by a single packet. Indeed, requests payload is usually shorter than the maximum length of the payload included in a network packet[2]. Responses are also composed by a single packet except in the picture case, whose response is composed by 38 fragments due to the

[2] This size is variable and it depends from the Maximum Transmission Unit (MTU) of the data-link layer, which assumes for instance the value of 1500 in case of an Ethernet connection.

large image size. In fact, the HTML page is sized 794 Bytes, the JPEG picture is sized 51874 Bytes, and the stylesheet is sized 91 Bytes. By analyzing the responses size, these values are never found, since each response also include protocol header information, such as the `Content-Length` value or the HTTP response code. Nevertheless, from the Connections Slots Table in Fig. 4 we can easily associate response sizes to the requested resources: HTML web page related response is sized 816 Bytes, CSS stylesheet related response is sized 432 Bytes, favicon related error response is sized 506 Bytes, and JPEG picture related response is sized 52169 Bytes.

B. *Malicious Traffic Analysis*

In this case we expect exactly $N_{slowloris}$ connections in the Connections Table. As reported in Fig. 5, this fact is confirmed in practice.

From the Connections Table we also see that a single request/connection slot is included in each connection. Indeed, since no response comes from the server, captured traffic only includes requests. Moreover, similarly to the Δ_{next} null results obtained in the legitimate case tests reported above, in this case we always have null values for the Δ_{delay}, Δ_{resp}, Δ_{next}, s_{resp}, and p_{resp} parameters. Analyzing Δ_{start} results for the attack we also notice that connections are almost always characterized by extremely similar values[3]. These facts confirm the potentialities of the proposed method to detect attacks targeting the application layer. Figure 5 also confirms our expectations on the Connections Slots Table cardinality, composed in this case by $N_{slowloris} = 10$ records. Moreover the Δ_{req} parameter assumes in this case an extremely high value compared to the legitimate case, as expected. In particular, this parameter always assumes a value near to $T_{last} = 540$ seconds. Due to additional times such as connection establishment, Δ_{req} value is strictly greater than T_{last}. This may represent a strange value, since we may expect a value near $T = 600$ seconds. In practice, if we consider a single connection/record, since capture has been interrupted after exactly 600 seconds, the last captured packet is registered at about $T_{last} = T - T_{WT} = 540$ seconds. Therefore, since we always have values strictly greater than T_{last}, the next sending phase would occur at $\Delta_{req} + T_{WT} > T$, thus not being registered by the capture.

[3] Note that displayed values are rounded to five decimals.

C_{id}	N_{req}	Δ_{start}
1	1	0.00030
2	1	0.00030
3	1	0.00307
4	1	0.00030
5	1	0.00030
6	1	0.00030
7	1	0.00030
8	1	0.00030
9	1	0.00030
10	1	0.00030

Connections Table

C_{id}	S_i	Δ_{req}	Δ_{delay}	Δ_{resp}	Δ_{next}	s_{req}	p_{req}	s_{resp}	p_{resp}
1	1	540.01366	–	–	–	309	11	–	–
2	1	540.01046	–	–	–	309	11	–	–
3	1	540.01486	–	–	–	309	11	–	–
4	1	540.01286	–	–	–	309	11	–	–
5	1	540.00911	–	–	–	309	11	–	–
6	1	540.01156	–	–	–	309	11	–	–
7	1	540.01109	–	–	–	309	11	–	–
8	1	540.01429	–	–	–	309	11	–	–
9	1	540.00979	–	–	–	309	11	–	–
10	1	540.01220	–	–	–	309	11	–	–

Connections Slots Table

Figure 5. Results for Slowloris Traffic

Another interesting result to notice is the p_{req} value. In this case, relatively to each connection/request, we may expect a value equal to $\frac{T}{T_{WT}} + 1 = 11$. In practice, due to the T_{WT} considerations made above, the last packet is never sent exactly at the last captured instant at time T, but later. Because of this, we may expect $p_{req} = \frac{T}{T_{WT}} = 10$.

Instead, by analyzing the results, during the capture time the attack sends 11 packets. This is because the Slowloris attack tool works in two phases [6]: in the first phase the initial part of the request is sent normally; subsequently, a repeated phase is accomplished by sending an additional payload string to the server, thus making use of the Wait Timeout, waiting for its expiration, and repeating this phase again. The obtained results are compliant to this behavior: relatively to the same connection, captured traffic includes a first packet containing the initial part of the request followed by a packet including the additional payload. Then, for the first time, the Wait Timeout is used. Therefore, we have two initial packets sequentially sent just after the connection has been established, followed by a repeated and endless slow packet sending.

Particularly, if we consider time interval $t_1 \in [0, T_{WT})$ we analyze that 2 packets are sent by the attacker, while 3 packets are sent in time interval $t_2 \in [0, 2 \cdot T_{WT})$, and so on. In general, considering time interval $t_k \in [0, k \cdot T_{WT})$, with $k = \frac{T}{T_{WT}}$, $p'_{req} = k + 1$ packets are sent. Hence, due to previous assumptions, since the first timeout expiration actually never occurs at T_{WT}, but later, we have $p_{req} = p'_{req}$, thus confirming the obtained results.

6. Possible Model Application

The purpose of our proposed model is intrusion detection. In order to provide a complete overview of the introduced model we will now report an example of a possible application for detecting different Slow DoS Attacks. We have focused on three Slow DoS Attacks that, in our opinion, currently represent the most serious threats, namely: Slowloris [8], SlowDroid [12], and Slow Read [11].

We analyzed the behavior on an Apache2 web server by targeting it with the three attacks for a duration of 600 seconds. The server configuration is the same of tests reported in Sect. 5, but in this case the attacks are now executed to yield a DoS on the victim; this was obtained by establishing and keeping the minimum amount of connections needed to cause a DoS. Attacks results have been compared to a legitimate situation including 30 minutes of network traffic related to different clients communicating with the victim's web server.

A. Data Analysis

By adopting the proposed model we have been able to extrapolate information potentially capable of detecting the executed threats. We have chosen to focus on Δ parameters. Indeed, although the proposed framework provides the ability to retrieve several parameters, for performance reasons, by accurately choosing and extrapolating only selected metrics, IDS performance are enhanced. Starting from the retrieved information, for each attack we analyzed results for Δ parameters, obtaining average and variance values. Results are shown in Table I. We have observed that for all the Slow DoS Attacks the required connections amount is successfully established (and maintained) with the server, thus achieving the primary goal of these threat. Notice that the success of the attacks is not relevant for our purpose, since our aim is to detect the attacks execution independently of their effectiveness.

TABLE I. Obtained Results Applying the Proposed Model to Different Attacks Targeting an Apache 2.2.22 Web Server

		Legitimate	Slowloris	SlowDroid	Slow Read
Δ_{start}	μ	0.60772	0.02750	0.02717	0.11781
	σ	1.71342	0.16084	0.16070	0.56281
Δ_{req}	μ	0.00426	**424.28700**	**549.21500**	0.01245
	σ	0.05230	132.62400	0.93285	0.08804
Δ_{delay}	μ	0.04966	–	–	**28.34780**
	σ	0.21568	–	–	65.39140
Δ_{resp}	μ	0.02266	–	–	0.00292
	σ	0.20359	–	–	0.04440
Δ_{next}	μ	2.30840	–	–	–
	σ	1.96204	–	–	–

Obtained Δ_{start} values are similar for all traffic conditions: indeed, the analyzed threats work by sending the first request (or part of the request) a few instants after a connection has been established. Analyzing results for the attacks it is clear that both Slowloris and SlowDroid may be detected by analyzing the Δ_{req} parameter, while Slow Read may reveal an anomaly relative to the Δ_{delay} parameter.

B. Legitimate Traffic Characterization

The above displayed table shows that attack results are different from legitimate results. Nevertheless, in order to properly identify an anomaly on the network a clear separation between legitimate and potentially anomalous traffic is needed.

It is indeed fundamental to establish a threshold beneath which two network scenarios can be considered semantically equivalent. A possible approach starts from results shown in Table I and defines legitimate traffic thresholds for the Δ parameters using the 3-sigma rule (also known as 68-95-99.7 rule) [42]. In statistics, this rule is used to identify the percentage of values which present characteristics similar to the mean value in a normal distribution with a width of one. In the empirical sciences field, the rule expresses a conventional heuristic that almost all the values are taken to lie within three standard deviations of the mean. In particular, assuming a Gaussian distribution of Δ parameters associated with legitimate traffics, all traffics associated to values $x \notin [\mu - 3\sigma, \mu + 3\sigma]$ are flagged as anomalous. This approach normally provides a confidence level equal to 99.7%.

In accordance to Table I, since $\mu - 3\sigma$ “left” thresholds always provide negative values, except for the Δ_{next} parameters, that can assume negative values, we only consider in general $\mu + 3\sigma$ as threshold, thus obtaining the following values: 5.74798 for Δ_{start} parameter, 0.16114 for Δ_{req} parameter, 0.69671 for Δ_{delay} parameter, 0.63344 for Δ_{resp} parameter, and -3.57772 (relatively to the left threshold) and 8.19452 (relatively to the right threshold) for Δ_{next} parameter.

Even if it was unexpected, the Δ_{start} threshold allows us to identify Slow Read as an anomaly, while the other attacks cannot be distinguished from legitimate traffic from this point of view. Nevertheless, by adopting the Δ_{req} threshold, both Slowloris and SlowDroid are detected from the system and flagged as anomalies.

The introduced detection system is therefore able to identify and detect all considered Slow DoS Attacks. In case a mixed traffic would be considered, including both legitimate and attack data, it should be clear that the success of the detection depends on the connections ratio relative to the attack, over the total amount of connections captured. Nevertheless, in order to perpetrate a successful attack, a large number of connections is usually established [5]. In addition, obtained results clearly show that the order of magnitude relative to legitimate and anomalous traffic is very different. Therefore, the proposed system should be considered an important tool to efficiently detect application layer attacks. Although a simple algorithm has been reported, the aim of the paper is not to introduce a novel Intrusion Detection System, but to provide instead a representation method able to extrapolate features characterizing application layer attacks.

7. Conclusions and Future Work

We defined and provided an innovative model aimed at detecting attacks targeting the application layer of the victim and working over the TCP transport protocol. An accurate selection and extrapolation of characteristic parameters is defined in the paper.

Thanks to our proposed model, it is possible to start from the introduced metrics (or a subset of them) and apply a particular intrusion detection method or algorithm, thus avoiding the characteristic features selection phase that researchers usually have to accomplish before applying their analysis algorithms. As a consequence, this approach provides a simplified and direct way to deploy an Intrusion Detection System. Moreover, since our model provides the ability to easily retrieve those parameters, testing of novel algorithms and consequently mitigation of such threats should be facilitated and accelerated.

Possible applications for our model may involve several research branches. For instance, it could apply to the extracted data of a statistical based intrusion detection or a machine learning study, trying to detect an anomaly through properties belonging to these specific fields. The model provides an infrastructure to researchers who want to design, adapt and adopt a particular algorithm for application layer menaces detection. In particular, by integrating our model with statistical analysis techniques it is possible to detect (even still unknown) network attacks acting at the application layer of the communication protocol stack. Therefore, the proposed work should be considered a fundamental resource for the research community, providing them an infrastructure able to identify and retrieve information characterizing network attacks acting at the application layer over the TCP protocol (i.e. Slowloris, SlowDroid, etc.), for a faster and more accurate deploying of the detection tool.

In order to show how the model behaves in a real scenario, we have accomplished accurate tests analyzing, describing, and comparing a legitimate situation and a malicious one, reporting and explaining in detail the obtained results and how data change in these two different scenarios. In this way, we have provided a real sample of the output generated by our tool and proved that it can be applied to different contexts, highlighting the potentialities of the proposed model to detect network attacks.

In order to provide an example of adoption of the proposed model we have also reported a simple intrusion detection system making use of our model to identify different attacks working at the application layer (e.g. slow denial of service, SQL injection attacks, etc.) targeting web servers. Since the model is not bounded to a specific detection algorithm or to a specific threshold definition, it is possible to integrate our work for defining novel detection systems, using approaches coming from different research areas (i.e., machine learning, statistics, neural networks, spectral analysis, game theory, etc.).

Although the proposed approach requires an observation of the protected server, a more accurate extrapolation of the selected features may be applied on the server itself, by analyzing sent and received data at the kernel level, rather than at the network level. Therefore, further work on the topic may involve more accurate retrieval operations and appropriate comparisons with the current approach.

References

[1] F. Gilham, R. Jagannathan, C. Jalali, P. G. Neumann. H. S. Javitz. A. Valdes. T. D. Garvey. T. F. Lunt, A. Tamaru. *A real-time intrusion-detection expert system (IDES)*. SRI International, Computer Science Laboratory, 1992.

[2] S. Kumar, E. H. Spafford. A software architecture to support misuse intrusion detection. 1995.

[3] K. Ilgun, R. A. Kemmerer, P. A. Porras. State transition analysis: A rule-based intrusion detection approach. *Software Engineering, IEEE Transactions on*, 21(3):181–199, 1995.

[4] T. Lunt. Detecting intruders in computer systems. In Proceedings of the 1993 Conference on Auditing and Computer Technology, 1993.

[5] E. Cambiaso, G. Papaleo, G. Chiola, M. Aiello. Slow DoS attacks: definition and categorisation. *International Journal of Trust Management in Computing and Communications - In press article*, 1, 2013.

[6] M. Aiello E. Cambiaso, G. Papaleo. Taxonomy of slow dos attacks to web applications. In *Recent Trends in Computer Networks and Distributed Systems Security*, pages 195–204. Springer, 2012.

[7] A. Kuzmanovic, E. W. Knightly. Low-rate TCP-targeted denial of service attacks: the shrew vs. the mice and elephants. *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 75–86, 2003.

[8] L. C. Giralte, C. Conde, I. M. De Diego, E. Cabello. Detecting denial of service by modelling web-server behaviour. *Computers & Electrical Engineering*, 39(7):2252–2262, 2013.

[9] H. Gonzalez, M. A. Gosselin-Lavigne, N. Stakhanova, A. A. Ghorbani. The Impact of Application-Layer Denial-of-Service Attacks. *Case Studies in Secure Computing: Achievements and Trends*, page 261, 2014.

[10] Microsoft-TechNet-Library. Microsoft Security Bulletin MS15-034 - Available at https://technet.microsoft.com/en-us/library/security/ ms15-034.aspx.

[11] S. Shekyan. Are you ready for slow reading? - Available at https:// community.qualys.com/blogs/securitylabs/2012/01/05/slow-read.

[12] E. Cambiaso, G. Papaleo, M. Aiello. SlowDroid: Turning a Smartphone into a Mobile Attack Vector. *2014 2nd International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 405–410, 2014.

[13] J. Heidemann, K. Obraczka, J. Touch. Modeling the performance of HTTP over several transport protocols. *Networking, IEEE/ACM Transactions on*, 5(5):616–630, 1997.

[14] W. Lee, S. J. Stolfo, K. W. Mok. A data mining framework for building intrusion detection models. *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*, pages 120–132, 1999.

[15] C.V.Zhou,C.Leckie,S.Karunasekera.Asurveyofcoordinatedattacks and collaborative intrusion detection. *Computers & Security*, 29(1):124– 140, 2010.

[16] S. R. Snapp, J. Brentano, G. V. Dias, T. L. Goan, L. T. Heberlein, C. L. Ho, K. N. Levitt, B. Mukherjee, S. E. Smaha, T. Grance. DIDS (distributed intrusion detection system)-motivation, architecture, and an early prototype. *Proceedings of the 14th national computer security conference*, pages 167–176, 1991.

[17] R. A. Kemmerer. NSTAT: a model-based real-time network intrusion detection system. Computer Science Department, University of California, Santa Barbara, Report TRCS97-18, http://www. cs. ucsb. edu/TRs/TRCS97-18. html, 1997.

[18] P. A. Porras, P. G. Neumann. EMERALD: Event monitoring enabling response to anomalous live disturbances. *Proceedings of the 20th national information systems security conference*, pages 353–365, 1997.

[19] J. Li, D.-Y. Lim, K. Sollins. Dependency-based distributed intrusion detection. *Proc. of DETER*, 2007.

[20] M. E. Locasto, J. J. Parekh, S. Stolfo, V. Misra. Collaborative distributed intrusion detection. 2004.

[21] D. Dash, B. Kveton, J. M. Agosta, E. Schooler, J. Chandrashekar, A. Bachrach, A. Newman. When gossip is good: Distributed probabilistic inference for detection of slow network intrusions. *Proceedings of the national conference on Artificial Intelligence*, 21:1115, 2006.

[22] H. Sedjelmaci, S. M. Senouci, M. Feham. An efficient intrusion detection framework in cluster based wireless sensor networks. *Security and Communication Networks*, 2013.

[23] T. Eswari, V. Vanitha. A novel rule based intrusion detection framework for Wireless Sensor Networks. *Information Communication and Embedded Systems (ICICES), 2013 International Conference on*, pages 1019–1022, 2013.

[24] S. Mutly, G. Yilmaz. A distributed cooperative trust based intrusion detection framework for MANETs. *ICNS 2011, The Seventh International Conference on Networking and Services*, pages 292–298, 2011.

[25] N. Komninos, C. Douligeris. LIDF: Layered intrusion detection framework for ad-hoc networks. *Ad Hoc Networks*, 7(1):171–182, 2009.

[26] N. Srivastav, R. K. Challa. Novel intrusion detection system integrating layered framework with neural network. *Advance Computing Conference (IACC), 2013 IEEE 3rd International*, pages 682–689, 2013.

[27] R. M. Elbasiony, E. A. Sallam, T. E. Eltobely, M. M. Fahmy. A hybrid network intrusion detection framework based on random forests and weighted k-means. *Ain Shams Engineering Journal*, 2013.

[28] S. Mukkamala, A. H. Sung. Identifying significant features for network forensic analysis using artificial intelligent techniques. *International Journal of digital evidence*, 1(4):1–17, 2003.

[29] K.-C. Lee, Z.-J. Hsu, L. Liu. Efficient Statistics Based Framework for Network Intrusion Detection. 2013.

[30] M. Almgren, U. Lindqvist. Application-integrated data collection for security monitoring. *Recent Advances in Intrusion Detection*, pages 22– 36, 2001.

[31] Y. Xuan, I. Shin, M. T. Thai, T. Znati. Detecting application denial-of-service attacks: A group-testing-based approach. *Parallel and Distributed Systems, IEEE Transactions on*, 21(8):1203–1216, 2010.

[32] M. Aiello, G. Papaleo, E. Cambiaso. SlowReq: A Weapon for Cyber-warfare Operations. Characteristics, limits, performance, remediations. *6th International Conference on Computational Intelligence in Security for Information Systems*, 2013.

[33] R. P. Kumar, J. Babu, T. G. Sekhar, S. B. Bhushan, D. Shamki, A. Al-Arussi, S. Tamrakar, M. Aloney, M. Kandpal, D. Sah. Mitigating Application DDoS Attacks using Random Port Hopping Technique. 2014.

[34] T. Siva, E. P. Krishna. Controlling various network based ADoS Attacks in cloud computing environment: By Using Port Hopping Technique. *International Journal of Engineering Trends and Technology (IJETT)*, 4, 2013.

[35] S. Y. Nam, S. Djuraev. Defending HTTP Web Servers against DDoS Attacks through Busy Period-based Attack Flow Detection. *KSII Transactions on Internet and Information Systems (TIIS)*, 8(7):2512– 2531, 2014.

[36] G. Macia-Fernandez, R. A. Rodriguez-Gomez, J. u. E. Diaz-Verdejo. Defense techniques for low-rate DoS attacks against application servers. *Computer Networks*, 54(15):2711–2727, 2010.

[37] Z. Wu, M. Yue, D. Li, K. Xie. SEDPbased detection of lowrate DoS attacks. *International Journal of Communication Systems*, 2014.

[38] M. Kiani, A. Clark, G. Mohay. Evaluation of anomaly based character distribution models in the detection of SQL injection attacks. Availability, Reliability and Security, 2008. ARES 08. Third International Conference on, pages 47–55, 2008.

[39] cPanel. How To Mitigate Slowloris Attacks - Available at http://docs.cpanel.net/twiki/bin/view/EasyApache/Apache/SlowlorisAttacks.

[40] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee. RFC 2616, Hypertext Transfer Protocol – HTTP/1.1 - Available at http://www.rfc.net/rfc2616.html.

[41] E. Cambiaso, G. Papaleo, G. Chiola, M. Aiello. Designing and Modeling the Slow Next DoS Attack. *CISIS 2015, International Conference*, pages 249-259. Springer International Publishing, 2015.

[42] M. Aiello, E. Cambiaso, S. Scaglione, G. Papaleo. A Similarity Based Approach for Application DoS Attacks Detection. *The Eighteenth IEEE Symposium on Computers and Communications*, 2013.

Enrico Cambiaso graduated in Computer Science at the University of Genoa, Italy, in 2012, with a thesis entitled "Analysis of slow DoS attacks". He is a PhD student at the University of Genoa and he collaborates with the Research National Council of Italy, working to the slow DoS field. His scientific interests are related to computer and network security, intrusion detection systems, covert channels and cloud computing.

Gianluca Papaleo graduated in Computer Science at the University of Genoa, Italy in 2005. He is a Fellow Researcher of the National Research Council since 2006. His main scientific interests are in the field of computer and network security, intrusion detection systems, wireless communications and covert channels. His current teaching activity in University of Genoa are focused on Wi-Fi security and tunneling protocols.

Giovanni Chiola graduated in Electronics Engineering at the Polytechnic of Turin, Italy in 1983. He was an Assistant Professor and subsequently an Associate Professor of Computer Science from 1985 to 1993 at the University of Turin, Italy. Since 1994, he is a Full Professor of Computer Science at the University of Genoa, Italy. In the past, his main scientific contributions have been in the fields of distributed simulation, performance modelling and evaluation of distributed systems, and stochastic Petri nets. His current research and teaching activities are focused on operating systems, distributed systems, peer-to-peer and computer and network security.

Maurizio Aiello graduated in 1994, worked as a free-lance consultant both for universities and research centre and for private industries. From August 2001, he is responsible of CNR network infrastructure. He is a Teacher at the University of Genoa and University College of Dublin; students Coordinator, fellowships and EU projects in the computer security field. His research are on network security and protocols.