Received: 5 April 2019 | Revised: 17 August 2019 | Accepted: 28 August 2019

DOI: 10.4218/etrij.2019-0164

ORIGINAL ARTICLE

ETRI Journal WILEY

Evaluating the web-application resiliency to business-layer DoS attacks

Mitra Alidoosti[1] | Alireza Nowroozi[1] | Ahmad Nickabadi[2]

[1]Malek-e-Ashtar University of Technology, Tehran, Iran
[2]Amirkabir University of Technology, Tehran, Iran

Correspondence
Alireza Nowroozi, Malek-e-Ashtar University of Technology, Tehran, Iran.
Email: alidoosti@mut.ac.ir

A denial-of-service (DoS) attack is a serious attack that targets web applications. According to Imperva, DoS attacks in the application layer comprise 60% of all the DoS attacks. Nowadays, attacks have grown into application- and business-layer attacks, and vulnerability-analysis tools are unable to detect business-layer vulnerabilities (logic-related vulnerabilities). This paper presents the business-layer dynamic application security tester (BLDAST) as a dynamic, black-box vulnerability-analysis approach to identify the business-logic vulnerabilities of a web application against DoS attacks. BLDAST evaluates the resiliency of web applications by detecting vulnerable business processes. The evaluation of six widely used web applications shows that BLDAST can detect the vulnerabilities with 100% accuracy. BLDAST detected 30 vulnerabilities in the selected web applications; more than half of the detected vulnerabilities were new and unknown. Furthermore, the precision of BLDAST for detecting the business processes is shown to be 94%, while the generated user navigation graph is improved by 62.8% because of the detection of similar web pages.

KEYWORDS
black-box testing, business layer, business process, denial-of-service (DoS) attack, logic vulnerability, web-application security

1 | INTRODUCTION

Web applications are one of the simplest ways to provide services to users. The ever-increasing demand for web applications has made their security a popular issue. According to Verizon, approximately 43% of the security-related events in 2017 were classified as attacks to web applications [1]. Web-security vulnerabilities constitute most of the reported vulnerabilities in the CVE database [2]. According to the Internet Security Threat Report issued by Symantec [3], in 2016, approximately 75% of the web applications were vulnerable. In the Trustwave report [4], business-logic vulnerabilities have been mentioned as the second most important vulnerabilities, thereby requiring the attention of researchers.

Business-logic vulnerabilities are susceptible to be misused, thereby causing denial-of-service (DoS) attacks. This kind of attacks is known as business-layer DoS attacks. DoS attacks are on the rise and constitute one of the most common attacks that target computer networks. DoS attacks are still considered a major threat [5,6]. This attack may occur in different layers of the OSI model, such as the application layer or the business layer. Over the past years, DoS attacks have moved from the lower layers of the OSI model to the upper layers and the application layer. However, nowadays, attacks have moved to the business layer, which lies even above the topmost layer of the OSI model. The attacks in the business layer are more effective and can be applied more easily. Detecting a business-layer DoS attack is much difficult because of the lower attack traffic [7]. DoS attacks in the business layer waste away the victim's resources. Typically, an attacker inflicts unusual computational overheads on the victim's resources, such as the CPU and memory, by sending one or a few more requests. However, DoS attacks in the business layer do not have malicious and bulky traffic. In fact, the web application vulnerable to business-layer DoS attacks has its normal functionality. The main reason for these vulnerabilities is the design flaw. These vulnerabilities are not an outcome of programming errors.

Business-layer DoS vulnerabilities are application specific and their detection requires understanding the logic of web applications. These vulnerabilities provide the possibility to the attacker to misuse the expected behavior of the web application. Therefore, detecting such vulnerabilities is difficult, and even web scanners are unable to detect them because of their failure to understand the logic of web applications. Therefore, the automatic detection of these vulnerabilities is not possible. The only way to detect them is using penetration testers, which depend on their skills and knowledge of the business processes of web applications [8–12].

Business-layer DoS vulnerabilities have been analyzed in numerous studies under different names (sophisticated DoS vulnerability [13], CPU-exhaustion vulnerability [13], algorithmic-complexity vulnerability [14], complexity vulner-ability [14], worst-case time/space complexity [14,15], ReDoS [14], second-order DoS vulnerability [16], infinite loop [17–19], semantic resource-exhaustion vulnerability [20], and tainted-loop vulnerability [20], high-complexity control structures [20]). Each of these studies analyzed specific cases of this type of vulnerability. The vulnerabilities examined in studies [13–20] are similar, as they may be exploited by limited legal requests. Furthermore, identifying such vulnerabilities would be easier by understanding the logic of the web application. Unfortunately, the approaches proposed to detect these vulnerabilities [16,17,20] have false positives and require the web-application source code for performing the vulnerability analysis. Moreover, these approaches are for the web applications written in a particular language.

1.1 | Analysis gaps

There is no dynamic approach for detecting business-layer DoS attacks. Besides, the previously proposed approaches were static and also had false positives. They were language dependent and, thus, specific to a particular language. None of them used the logic of web applications to detect vulnerabilities, although knowing the logic of the web application would make the identification of logic vulnerabilities easier.

In this study, we propose the business-layer dynamic application security tester (BLDAST), which is a black-box technique to assess the DoS resiliency. We present a dynamic approach to detect the web applications vulnerable to business-layer DoS attacks. Our approach does not have any false positives and is not language dependent either. Our previous work, BLProM [21], was a black-box technique to detect the business layer of a web application. It extracted business processes from a web application. This study extends and improves BLProM to detect the business processes in a web application. BLDAST uses improved BLProM to extract the business processes from a web application and finally identifies the vulnerability of a web application to business-layer DoS attacks. Our test scenarios are context aware, implying that they are business aware and understand the business logic of web applications. The proposed method is independent of the technology used in the web applications and automatically detects the vulnerabilities. Besides, it is shown that the precision of BLDAST in identifying the business-logic processes is approximately 94%. Furthermore, BLDAST detects similar web pages to generate the optimal user navigation graph and to prevent the generation of an infinite graph. The BLDAST-generated user navigation graph has improved by approximately 62.8%.

In short, in this study, we offer the following novelties.

- We define business-layer DoS attacks.
- We present BLDAST, a black-box, dynamic technique for security testing of web applications, to identify the business-logic vulnerabilities against DoS attacks.
- We introduce a new black-box approach to identify similar web pages and business processes in the web applications.
- The precision of BLDAST in detecting the business processes is approximately 94%, while the generated user navigation graph is improved by approximately 62.8%.

The rest of this paper is organized as follows. Section 2 presents the related works. Section 3 presents the definition of business-layer DoS attacks. Section 4 describes the proposed approach for the black-box, dynamic testing of web applications. Sections 5 and 6 show the implementation and evaluation of the proposed approach, respectively. Finally, Section 7 concludes this study.

2 | RELATED WORKS

2.1 | Logic attacks (Business-Logic Attack)

There are two methods to prevent logic attacks (business-layer attacks): (i) the defense method (detecting the attacks during the execution time) and (ii) the prevention method (detecting the logic vulnerabilities existing in the web application). BLOCK [22] has used the defense method to prevent logic attacks. It first obtains the behavioral model of the web application by monitoring the interaction of clients and the web application, and it then extracts a set of constants from the sequence of requests/responses and session variables. Subsequently, it uses the extracted constants to evaluate the requests/responses. BLOCK detects as an attack each request/response that may violate the known constants.

SENTINEL [23], Doupe [24], Pelegrinio [25], and DetLogic [26] have used black-box prevention methods to detect logic vulnerabilities. SENTINEL [23] detected the logic shortcomings of accessing the database. It modeled a web application as a state machine. The black-box method was used to extract the features of the web application and detect the queries violating the extracted features. Web-application features include the constants extracted from the SQL queries and session variables. Each malicious query that violates the detected constants is detected as an attack.

Doupe and others [24] have presented an approach for detecting the internal state of a web application by observing its output. The proposed approach crawls different states of the web application and detects XSS and SQL injection vulnerabilities by applying proper input vectors. In fact, active method is used to detect vulnerabilities.

Pelegrino and Balzarotti [25] automatically extracted a number of behavioral patterns of users' traffic. First, a model of web application was extracted, and then attack vectors were applied to it. The stored traffic of the user is analyzed to detect the patterns related to the logic of the web application. Such behavioral patterns should be verified by the extracted model.

DetLogic [26] was an approach for detecting different types of logical vulnerabilities such as parameter manipulation, access control, and workflow bypass. It models the expected behavior of the web application as a state machine, Subsequently, it extracts the limitations of the state machine and employs the detected constraints to create an attack vector for detecting the aforementioned vulnerabilities.

2.2 | Defense against application-layer DoS attacks

Application-layer DoS attacks usually render the web application irresponsive. Attackers turn the system off by sending malicious inputs [21] and put the system in an endless loop or result in a recursive recall with superlinear complexities [25,26]. A dynamic analysis to detect application-layer DoS attacks generates inputs that result in the worst execution time, and, therefore, the application enters an endless loop [22–24]. A dynamic analysis of large web applications is difficult, and, in some cases, the inputs with the worst execution time cannot be generated [12].

SLOWFUZZ [14] provided a framework for the automatic identification of algorithmic-complexity vulnerabilities (algorithms running at the worst time and causing DoS attacks). It automatically detected the entries that caused the worst behavior (high-order times) in the program by checking the amount of resources consumed in the program.

Looper [18] was an approach for performing the dynamic analysis of a web application by determining whether the application was endless.

WISE [15] generated the inputs that determined the execution time of a web application. It presented an automatic approach for detecting the efficiency problems of a web application. In a web application with no constraints on the length of inputs, WISE was able to detect the input with the worst execution time.

Gupta [21] presented a dynamic approach to prove whether the executed application was in endless execution.

A static analysis to identify the web applications vulnerable to application-layer DoS attacks specifies the pieces of program codes that depend on the user input and have high complexity. By using these codes, the attacker causes the program to be irresponsive [17].

SAFER [19] was a static-analysis tool for detecting the web applications vulnerable to DoS attacks before the application deployment. SAFER focused on the complicated DoS attacks that result in the loss of resources in network-oriented applications. The attacker often transmits one or a few number of requests for increasing the computational load of the internal resources of a system such as processor or stack space.

TORPEDO [16] was a static-analysis tool for detecting a second-order DoS attack in web applications. In this type of attacks, first the database was filled with a great amount of data. Second, costly operations were performed on the input data, leading to the depletion of resources. However, TORPEDO was able to detect only the vulnerability of web applications written in PHP and had false positives.

3 | BUSINESS-LAYER DOS ATTACK

In this part, the business-layer DoS attack is defined, and then an example of business-layer DoS vulnerabilities is presented.

3.1 | Definition 1 (Business-Layer DoS Attack)

A business-layer DoS attack is a business-aware sequence of requests compromising the system's availability and depleting its resources (bandwidth, CPU, memory, etc.) by exploiting a developmental or operational software flaw/fault in the business-layer services of the web application.

3.2 | Example of Business-Layer DoS Attack

Assume that an e-commerce web application provides the facility for its users to comment on a specific product and then shows all the comments for that product. If the application considers no constraints on the number of submitted comments such that comments can be submitted infinitely, a business-layer DoS attack can be applied to the application by submitting a large number of comments and viewing all the submitted comments. In such attacks, by sending a few numbers of requests in parallel to view the submitted comments, the web application would be unable to respond.

In the above-mentioned example, if the web application uses mechanisms such as CAPTCHA before a user submits comments in the database, it would not be possible to automatically submit a large number of comments. In addition, for viewing the comments section, the application must investigate the submitted comments. If there are too many comments, then not all but a fraction of them are shown. In this case, the web application is not vulnerable to a business-layer DoS attack.

4 | BUSINESS-LAYER DYNAMIC APPLICATION SECURITY TESTER

BLDAST evaluates the resiliency of a web application to business-layer DoS attacks. Figure 1 depicts the BLDAST overview. BLDAST is situated as a proxy between the web application and the web server. BLDAST comprises the following four main steps:

1. Extracting the user navigation graph;
2. Detecting the business processes of the web application;

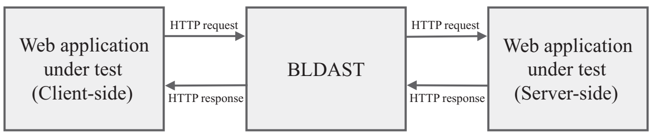
Web application under test (Client-side)
HTTP request
HTTP response
BLDAST
HTTP request
HTTP response
Web application under test (Server-side)

FIGURE 1 BLDAST overview

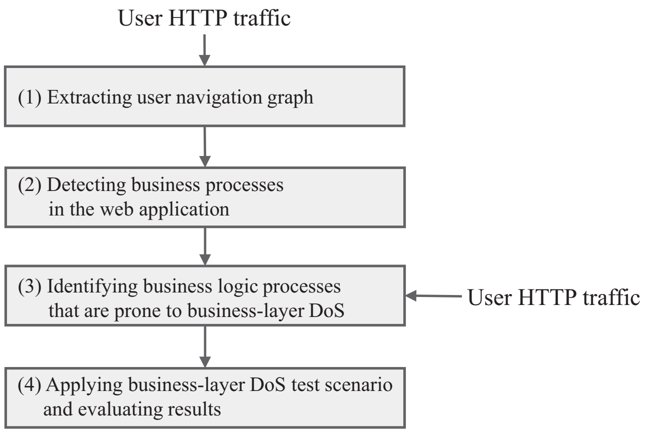
User HTTP traffic
(1) Extracting user navigation graph
(2) Detecting business processes in the web application
(3) Identifying business logic processes that are prone to business-layer DoS
User HTTP traffic
(4) Applying business-layer DoS test scenario and evaluating results

FIGURE 2 BLDAST architecture

3. Identifying the business processes that are vulnerable to business-layer DoS attacks;
4. Applying business-layer DoS test scenarios and evaluating the results.

First, a normal user crawls the web application, following which his/her HTTP traffic is stored. BLDAST first extracts the user navigation graph from the stored traffic. Then, it uses the generated graph to extract the business processes of the application. It detects the business processes vulnerable to DoS attacks, and it finally applies business-layer DoS test scenarios according to the detected critical processes. Figure 2 depicts the proposed steps for the dynamic security testing of web applications in the business layer.

4.1 | Extracting user navigation graph

BLDAST first extracts the user navigation graph from the stored traffic. Figure 3 depicts the steps involved in

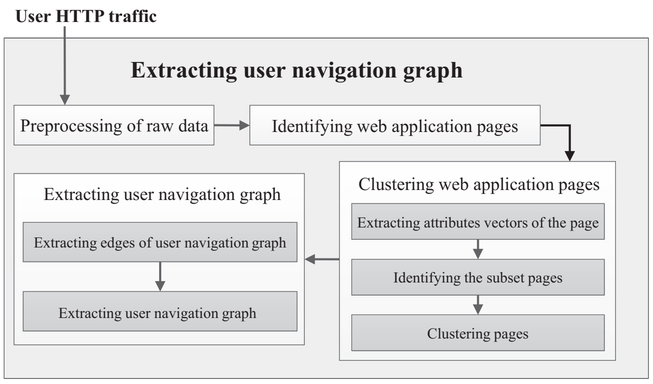
User HTTP traffic
Extracting user navigation graph
Preprocessing of raw data
Identifying web application pages
Extracting user navigation graph
Extracting edges of user navigation graph
Extracting user navigation graph
Clustering web application pages
Extracting attributes vectors of the page
Identifying the subset pages
Clustering pages

FIGURE 3 Extracting the user navigation graph

extracting the user navigation graph. The steps are as follows:

1. Preprocessing the raw input data;
2. Detecting the web pages of the web application existing in the stored traffic;
3. Clustering the web pages;
4. Extracting the user navigation graph.

4.1.1 | Preprocessing raw input data

BLDAST sanitizes data to eliminate unnecessary items. In this study, only HTTP requests and HTTP responses are needed; nonetheless, the HTTP responses having a successful status code 200 are needed. BLDAST eliminates the responses having unsuccessful codes and their corresponding requests. In addition, in this study, only the GET and POST requests are needed.

4.1.2 | Detecting web-application pages existing in the stored traffic

Each web page of the web application can be modeled as a pair (request set, response set). The request set is a set of HTTP requests sent for loading the page. The response set is a set of HTTP responses sent for loading the page.

A user's stored traffic contains the main HTTP requests for loading the webpage and also secondary requests meant to load files of the page. To model the pages existing in the HTTP traffic as a pair (request set, response set), first the main requests in the HTTP traffic are detected. The HTTP traffic based on the main detected requests is then divided into blocks. Each block includes the main request and the HTTP traffic between the two main requests. All the requests and responses in the same block represent the request and response sets of a page. BLDAST considers a request as the main request if its referer header is different from the referer of the next request. In addition, the first request existing in the traffic is considered the main request because the first traffic does not have a referer. It should be mentioned that the referer header is a header of the HTTP request that represents the URI address of the previous page visited by the user.

Thus far, web-application pages have been modeled as a pair (request set, response set). The main response in the response set of each page should also be detected. The main responses have a content-type header with a text/html value. The main response is of text/html type. Furthermore, if the last response of the traffic is the main response, then the last HTTP request will be considered the main request. Therefore, each page of the web application is modeled as a pair (request set, response set), and the main requests and responses are marked.

4.1.3 | Clustering web pages

In this step, BLDAST clusters web the pages. The purpose of clustering is to place similar web pages in one cluster. Each cluster represents one node of the user navigation graph. Therefore, the infinite development of the graph is avoided.

Each pair (request set, response set) shows one page of the web application. To extract the optimal user navigation graph, similar pages should be identified and subsequently clustered. In the user navigation graph, the nodes represent the unique pages of the web application and the edges the connection between the pages. In our context, two pages are considered similar when the user can perform similar actions on them. For instance, consider two pages that have a button. The button on the first page is "continue," and the button on the second page is "save." These two pages are different because the user performs different actions on them.

Because the final goal of BLDAST is to detect vulnerabilities, clustering identifies similar pages based on the goal of BLDAST. The input fields and hyperlinks that provide the possibility of interaction with the application are critical points in detecting vulnerabilities. Therefore, HTML elements that have critical points are considered for detecting similar pages. These elements include buttons, inputs, and anchors existing in each page. In addition, in the web-application pages, the position of images is related to the logic of the page. Therefore, the position of image is another important element in detecting similar pages.

4.2 | Definition 2-1 (Similar Pages)

Two pages are considered similar if the user is able to perform the same actions on them and if the positions of important HTML elements in the pages are also the same. Important HTML elements include buttons, inputs, anchors, and images existing in each page.

4.3 | Definition 2-2 (Similar Pages)

Pages whose structures are the subsets of another page in terms of important HTML elements are similar to the reference page.

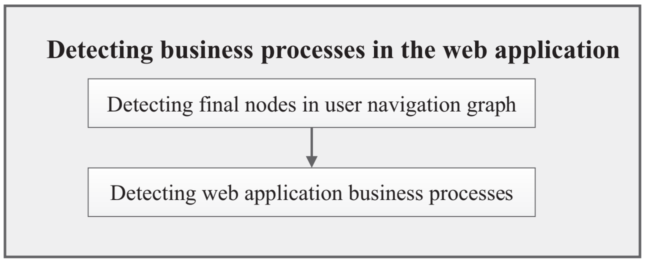
Detecting business processes in the web application
Detecting final nodes in user navigation graph
Detecting web application business processes

FIGURE 4 Detecting business processes in web applications

BLDATS identifies similar pages by comparing the feature vectors of two pages with each other. The feature vector of each page is the DOM path of important HTML elements existing in the page. BLDAST considers every two pages with the same feature vector as similar pages and puts them in one cluster. The algorithm used to identify similar pages is described in Section 5.1.

4.3.1 | Extracting user navigation graph

In this step, BLDAST first extracts the graph edges that connect clusters. Each cluster represents a unique web-application page. Each cluster has a set of similar pages, each of which contains a URI and a referer field. Therefore, each cluster has a set of URIs and a set of referers, both of which are associated with the pages existing in that particular cluster. To find graph edges, the URI set of each cluster is compared to the referer set of other clusters. If they have at least one member in common, they will connect to each other. Assume that the URI set of cluster C_1 shares members with the referer set of clusters C_2 and C_3, then one can move from C_1 to C_2 and C_3 ($C_1 \rightarrow C_2$ and $C_1 \rightarrow C_3$); in other words, C_1C_2 and C_1C_3 are graph edges.

4.3.2 | Definition 3 (User Navigation Graph)

This graph is represented by tuple $<C_0, C, E>$. Here, C denotes the set of graph nodes, $C_0 \subseteq C$ the initial (first) node of the graph, and $E \subseteq C \times C$ the set of graph edges.

4.4 | Detecting business processes in web application

In this step, BLDAST first identifies the final nodes in the user navigation graph, and it then detects business processes. The steps to detect business processes are depicted in Figure 4. We now define the web-application process, final node, and business process.

4.4.1 | Definition 4 (Web-Application Process)

A process P in a web application is a sequence of edges in the user navigation graph, such as $E_1, E_2, \ldots, E_k$, where $E_i \in E$ and $E_i = C_{i-1}C_i$.

4.4.2 | Definition 5 (Final Node in the User Navigation Graph)

The final node F is the node such that when a web application reaches it, then the business process is completed.

By investigating HTTP responses, the final node can be identified. For instance, when a product is purchased, an expression such as “thank you for your purchase” is shown. By identifying a set of such expressions and searching for these expressions in the response messages, the final node can be determined. The keywords used to identify the final node are thanks, congratulations, successfully, log off, search results, among others. In addition, some buttons are good indicators for identifying the final node. The page after save button, the page after create button, and the page after submit button are some examples in this context.

4.4.3 | Definition 6 (Business Process in the Web Application)

A business process in the web application is a process that meets at least one of the following requirements:

1. The beginning node of the process is the initial node of the user navigation graph (C_0), and the ending node of the process is the final node of the user navigation graph (F).
2. If the process passes the beginning node once again and if the process length is greater than 2. (If in a process, the user returns to the beginning node and if the length of the loop is greater than 2, it is a business process).

4.5 | Identifying business processes prone to business-layer DoS attack

BLDAST identifies business-logic vulnerabilities in a web application by analyzing the interaction of business processes with one another. BLDAST detects the business processes vulnerable to business-layer DoS attacks in two steps. In the first step, the processes that submit data to the database are identified. These processes should be repeatable for any desired number of times. These processes are called “inserting process.” In the second step, the processes that perform a costly operation on the inserted data are identified. These processes are called the "retrieving process." Both the steps are depicted in Figure 5. In other words, BLDAST identifies the parameters shared by business processes such that the first process is able to write on the mentioned parameters for any desired number of times and that the second process employs the values of the parameters.

The "inserting process" can be repeated for any desired number of times if the application does not have a defense mechanism such as CAPTCHA, Turing test, or similar mechanisms because these defense mechanisms prevent the automatic repetition of the process. The "retrieving process" can be performed if the application performs no sanitization on the data retrieved from the database (for example, bounding the size of the retrieved data).

4.5.1 | Definition 7 (Tainted Parameter)

Assume that a web application W includes a set of web pages, where each page is defined as a pair (request set, response set). Each request with either the GET or POST method might have parameter(s) and is denoted by PP. Notably, PP is called the "*tainted parameter*" if there is possibility of submitting as many as desired requests for unlimited times, and the value of PP is stored in a table of the database following each submitted request.

4.5.2 | Definition 8 (Inserting Process)

Assume that a web application W includes a set of business processes. Each business process further includes a set of web pages. If at least one of the business process's pages has a "*tainted parameter*," the business process is called "*inserting process*."

4.5.3 | Definition 9 (Response Time)

The response time of a page is the time between submitting the main HTTP request and receiving the main HTTP response. To calculate the response time, it would be sufficient to subtract the time of receiving the main response from the time of submitting the main request.

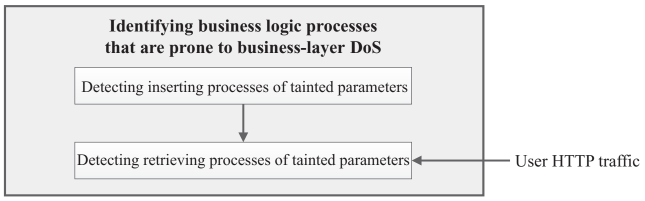
Identifying business logic processes that are prone to business-layer DoS
Detecting inserting processes of tainted parameters
Detecting retrieving processes of tainted parameters
User HTTP traffic

FIGURE 5 Identifying DoS-attack-prone business processes

The response time of a page is calculated twice, that is, before and after executing the "*inserting process*" for many times.

$$\text{ResponseTime} = \text{Time}_{\text{Receiving main response}} - \text{Time}_{\text{Submitting main request}}. \quad (1)$$

4.5.4 | Definition 10 (Critical Page)

Assume that a web application W includes a set of web pages. "*Critical page*" is a page that retrieves all the values submitted for the "*tainted parameter*" and employs them as the input. The response time of the "*critical page*" is a function of n, which is the number of values entered for the "*tainted parameter*" or the number of times the "*inserting process*" is executed. In other words, increasing the number of times the "*inserting process*" is executed would increase the response time of the "*critical page.*"

4.5.5 | Definition 11 (Retrieving Process)

Assume that a web application W has a set of business processes. Each business process includes a set of web pages. If at least one of the business process's page is a critical web page, the business process would be a "*retrieving process.*"

If the "*inserting process*" and the "*retrieving process*" exist in the web application, the web application would be vulnerable to business-layer DoS attacks.

Identifying "inserting process"

To identify the "inserting process" in the application, the pages with at least one parameter should be marked. Then, the processes with at least one marked process are selected. Now, it should be determined whether the process can be executed many times. In other words, running the process several times should not result in an error page. Moreover, the process should reach its final state.

The selected process is run time and again (for example, for 25 000 times). If the process reaches its final state in all the executions, it is considered an "inserting process."

Identifying "retrieving process"

After identifying the "inserting process," to identify the "retrieving process," it would be sufficient to crawl the web application again, store the HTTP traffic, and calculate the response time of the pages; the pages whose response time has increased significantly are identified as "critical pages." A threshold value must be considered for the response time. After investigating various applications, it is concluded that if the response time of the page has increased 10 times, the page can be considered a "critical page." The processes that have a "critical page" are considered "retrieving processes."

4.6 | Applying business-layer DoS test scenarios and evaluating results

The DoS test scenario in the business layer is performed after knowing about the "inserting process" and the "retrieving process." We call this scenario business-layer DoS (BLDoS). BLDoS comprises two steps. In the first step, the database is filled with a lot of unworthy data. In other words, BLDoS repeats the "inserting process" a lot of times. In the second step, it performs the "retrieving process" for a limited number of times in parallel. In fact, the second step performs a costly operation on the data entered in the database to consume system resources, thereby making the application inaccessible for a long time. By repeating the second step continuously, the application becomes inaccessible forever. Both these steps are depicted in Figure 6.

In the first step of BLDoS, where data flooding occurs, the "inserting process" is repeated in separate time intervals. In each interval, the "inserting process" is repeated for a few number of times until a lot of data enters the database. In this case, BLDoS cannot be detected using defense mechanisms, and as BLDoS occupies a small bandwidth, it cannot be detected using detection tools either.

As mentioned, there are the following two main conditions guaranteeing that the web application is vulnerable to business-layer DoS attacks:

1. The ability to add data into the database for any desired number of times.
2. The time order of retrieving the data increases in proportion to the increase in the number of entered data.

5 | IMPLEMENTATION

In this section, the details of implementing different BLDAST sections are described.

5.1 | Clustering similar pages

BLDAST clusters pages by detecting similar pages. The clustering comprises the following three main steps:

1. Extracting the feature vector of each web page;
2. Identifying similar pages;
3. Clustering the pages.

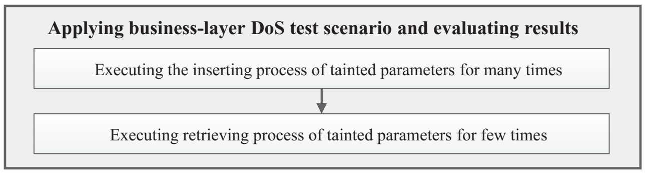
Applying business-layer DoS test scenario and evaluating results
Executing the inserting process of tainted parameters for many times
Executing retrieving process of tainted parameters for few times

FIGURE 6 Applying BLDoS

5.1.1 | Feature vectors of pages

BLDAST represents each page as a pair (request set, response set). In this step, BLDAST performs data mining on each pair to extract the corresponding feature vector of each page. BLDAST models each page using the following feature vectors:

WP = total web pages existing in an application,
$\forall w \in WP\ w = (DOM_{inputs}, DOM_{buttons}, DOM_{anchors}, DOM_{imgs})$,
$DOM_{inputs} = \Pi_i DOM\ (input_i)$,
$DOM_{buttons} = \Pi_i DOM\ (buttons_i)$,
$DOM_{anchors} = \Pi_i DOM\ (anchor_i)$,
$DOM_{imgs} = \Pi_i DOM\ (img_i)$.

1. DOM (input): DOM path of <input>tag in the page + value of type attribute in <input>tag + value of name attribute in <input>tag.
2. DOM (button): DOM path of the button existing in the page + text on the button.
3. DOM (anchor): DOM path of <a> tag in the page.
4. DOM (img): DOM path of the image existing in the page.

5.1.2 | Identifying similar pages

After extracting the feature vector of each page, similar pages must be identified. The pages whose feature vector are a subset of that of another page or have the same feature vector as that of another page are considered similar pages.

The feature vector of each page of the application has four elements. The feature vector of page 1 is similar to that of page 2 if the following conditions are met:

- All the elements in the feature vector of page 1 are equal to the corresponding elements in the feature vector of page 2.
- All the elements in the feature vector of page 1 are a subset of the corresponding elements in the feature vector of page 2 and vice versa.
- If one of the several elements in the feature vector of page 1 is a subset of the corresponding element in the feature vector of page 2, then the other elements of the feature vector of page 1 should be the same as the corresponding elements in the feature vector of page 2.
- A null element is a subset of any element.

6 | EVALUATION

To evaluate the proposed method, BLDAST was used in different web applications. Because BLDAST applies costly operations on the test applications, it cannot be applied to under-load and online applications. Thus, tests were performed on six open-source applications that could be loaded offline. Table 1 presents the web applications used for the evaluation.

Our test bed had a web server (test target) and a client (BLDAST system). The web server and the client were loaded on a virtual machine. The web server runs Windows 8.1 on a Pentium dual-core 2.20-GHZ CPU with 8 GB RAM. The BLDAST machine runs Windows 8.1 on an Intel core i7 2.20-GHZ processor with 8 GB RAM. Both the virtual machines run on Windows 7 with 1 GB RAM and are on the same local area network. Table 2 summarizes the test results on selected web applications. More than half of the identified vulnerabilities are unknown.

6.1 | Performance evaluations

To evaluate the performance of BLDAST, we define the following performance indicators.

- Accuracy: It denotes the precision of BLDAST in detecting vulnerabilities.

TABLE 1 Selected web applications for evaluation

Web applications	Description
WackoPicko	Photo sharing website [22–25]
osCommerce-2.3.4	E-commerce [22,25–27]
TomatoCart-1.1.8.6.1	E-commerce [12]
VirtueMart-3.0.14	E-commerce
OpenConf-6.8.1	Conference management system
Scarf-2007-02-27	Conference management system [22–25]

TABLE 2 Summary of experimental results

Web applications	#LOC	# Attack requests	# vulner-abilities (E)	# Vulner-abilities (D)	#TP	#FP
WackoPicko	4,037	11	3	3	3	0
osCommerce-2.3.4	86 693	1	5	5	5	0
TomatoCart-1.1.8.6.1	95 478	1	4	4	4	0
VirtueMart-3.0.14	93 421	9	1	1	1	0
OpenConf-6.8.1	22 889	11	13	13	13	0
Scarf-2007-02-27	1686	10	4	4	4	0

Abbreviations: LOC, lines of code; E, existing; D, detected; TP, true positive; FP, false positive.

- Time complexity: It is the time complexity of BLDAST in analyzing web applications.
- Overhead: It is the average CPU usage, memory usage, and bandwidth usage of the BLDAST system.
- Effectiveness: It signifies how long BLDAST causes web applications to become unresponsive.

To evaluate the accuracy of BLDAST, we used the following metrics: true positive rate (TPR), false positive rate (FPR), false negative rate (FNR), precision, and recall. The TPR, recall, and precision of BLDAST are 100%. The FPR and FNR of BLDAST are 0%, as they have no false positive or false negative in detecting vulnerabilities.

The time needed for BLDAST to detect a vulnerability depends upon the number of iterations of the inserting process. Its time complexity is O(n), where n denotes the number of items inserted in the database during phase 3. Phase 3 (identifying the business processes vulnerable to business-layer DoS attacks) is BLDAST's critical phase, as other phases (phases 1, 2, and 4) are executed very soon within reasonable time. In addition, BLDAST executes phase 3 in separate time intervals. In each interval, the "inserting process" is repeated for a few number of times until a lot of data enters the database. Thus, the exact running time of BLDAST cannot be calculated.

To calculate the overhead of BLDAST, we used some metrics such as memory usage, CPU usage, and bandwidth usage of the BLDAST system during the execution time for detecting the vulnerabilities of the selected web applications. The average CPU usage, average memory usage, and average bandwidth usage of the BLDAST system were 13.1%, 35.6%, and 604 KB/s, respectively.

To evaluate the effectiveness of BLDAST, we calculated the time for which the server became unresponsive. This time depends upon the number of times the insertion process is executed. Table 3 presents the results of applying the test scenarios to selected applications. On average, if the "inserting process" was repeated 25 000 times, the web application became inaccessible after 36 parallel executions of the "retrieving process." The average time during which the application was inaccessible after 36 parallel executions of the "retrieving process" was 440 s.

6.2 | Clustering evaluations

One of the main steps in extracting the user navigation graph is clustering the web pages. The accuracy and precision of the clustering operation are shown by using the following metrics:

- True Positive: samples that are fitted correctly in their cluster.
- False Positive: samples that are fitted incorrectly in the intent cluster.
- Precision: it is calculated as follows:

$$\text{Precision} = \frac{\text{TruePositive}}{\text{TruePositive} + \text{FalsePositive}}. \quad (2)$$

- Average running time: it shows the average running time of BLDAST spent on clustering the web pages. On average, BLDAT clusters web pages in 2 s.

Table 4 demonstrates the values of the mentioned metrics for clustering the selected web-application pages. The first column (#samples) shows the number of web pages in the HTTP traffic. The second column (#clusters) shows the number of clusters. In the subsequent columns, the previously mentioned measures are calculated for each web application. The table entry "Average" in the last row shows the average of the mentioned metrics for the selected web application. In Table 4, the average value of the clustering-precision metric for the selected applications is 0.95. On average, BLDAST performs clustering with precision of 0.95.

BLDAST extracts the user navigation graph for each selected web application. The results of the extraction are presented in Table 5, where the "# Graph Nodes" indicates the number of clusters obtained from the clustering operation. For instance, the user navigation graph for WackoPicko has 22 nodes, 48 edges, and 12 business processes.

Table 6 presents the characteristics of the user navigation graph without applying clustering. For instance, if clustering is not used to generate the user navigation graph, the generated user navigation graph for WackoPicko would have 89 nodes, 270 edges, and 48 business processes. It is clear that using clustering to identify similar pages results in the generation of the optimal graph.

TABLE 3 Average of results of applying the test scenarios to selected applications

#Execution of inserting process	#Parallel execution of retrieving process	Time the server is unresponsive (s)
25 000	36	440
50 000	17	636
75 000	7	808
100 000	1	More than 1 h

TABLE 4 Evaluation of the clustering of selected web-application pages

Metrics Web applications	# Samples	# Clusters	# True positive	# False positive	Clustering precision
WackoPicko	89	22	99	15	0.87
osCommerce	210	40	227	5	0.98
Tomatocart	150	66	180	7	0.96
Virtuemart	130	66	122	8	0.94
OpenConf	104	39	104	0	1.00
Scarf	42	18	41	1	0.98
Average	120.83	43.00	128.83	6.00	0.95

Table 7 presents the improvement in the BLDAST-generated user navigation graph compared with the pre-clustering graph. The last column, "Average," shows the improvement in the selected web applications. From Table 7, it can be observed that the number of nodes in the user navigation graph has improved by 62.8%. In other words, it means that the number of graph nodes in the output of BLDAST is 62.8% lower than the number of graph nodes without applying clustering.

6.3 | Evaluation of the proposed model

To evaluate the proposed model for identifying business processes, the following metrics are used. These measures are adopted from [28] with minor modifications. It should be mentioned that the model is the user navigation graph. The extracted business processes are the same processes that previously existed in the model. Log is the user HTTP traffic. The processes in the log are the same processes that the user has gone through and whose report is available in the HTTP traffic.

6.3.1 | Fitness

Fitness shows how many of the processes existing in the log are included in the proposed model. In other words, it describes the ability of the proposed model to numerically generate the processes existing in the log. The following formula calculates fitness:

$$F = \frac{\#\,\text{business processes common in both the log and the model}}{\#\,\text{business processes in the log}}. \quad (3)$$

The numerator denotes the number of processes in the proposed model, which are available in the log as well. The denominator denotes the number of processes available only in the log. The value of fitness is between zero and one, and the closer it is to one, the better the fitness would be. In our user navigation graph, the graph nodes denote unique pages. Each node might represent several similar pages. Therefore, each process existing in the graph might represent several processes in the log.

6.3.2 | Simplicity

Simplicity calculates the complexity of the proposed model. It is calculated by comparing the number of processes in the proposed model with the number of processes existing in the log. The number of processes in the model is the main factor of model complexity and model error. If each process exists only once in the model, the proposed model is considered a simple model. Considering the above discussion, simplicity is calculated as (4).

TABLE 5 Characteristics of user navigation graph generated using BLDAST for selected web applications

Web applications metrics	WackoPicko	osCommerce	TomatoCart	VirtueMart	Openconf	Scarf
# Graph nodes	22	40	66	66	39	18
# Graph edges	48	66	87	87	39	43
# Business-logic process	12	23	31	31	42	18

TABLE 6 Characteristics of user navigation graph for selected web application without applying clustering

Web applications metrics	WackoPicko	osCommerce	TomatoCart	VirtueMart	Openconf	Scarf
# Graph nodes	89	210	150	130	104	42
# Graph edges	270	379	410	398	426	98
# Business-logic process	48	57	39	47	96	29

$$S=1-\frac{\#\text{business processes ignored by the model }+\#\text{duplicate business processes in the model}}{\#\text{ business processes in the model}+\#\text{business processes in the log}}. \tag{4}$$

6.3.3 | Precision of the model

Precision is the ratio of the number of processes in the model that do not exist in the log to the total number of processes in the log.

$$P=1-\frac{\#\text{business processes in the model that do not exist in the log}}{\#\text{business processes in the log}}. \tag{5}$$

Table 8 presents the evaluation results of the proposed model in identifying business processes for the selected web applications. For instance, the fitness of our proposed model is 0.94 on average. The average simplicity and precision of the proposed model are, respectively, 0.97 and 0.94.

7 | CONCLUSION AND FUTURE WORK

In this study, an approach known as BLDAST was proposed to evaluate the robustness of web applications against business-layer DoS attacks. BLDAST comprises the following four steps: extracting the user navigation graph, finding the business processes of application, identifying business-layer processes vulnerable to DoS attacks, and applying DoS test scenarios against web applications in the business layer. On the laboratory scale, BLDAST was used to evaluate the robustness of six web applications against business-layer DoS attacks. More than half of the detected vulnerabilities were new and

TABLE 7 Improvement of graph generation

Web applications metrics	WackoPicko	osCommerce	TomatoCart	VirtueMart	Openconf	Scarf	Average
# Graph nodes	75.3	76.5	56	49.2	62.5	57.1	62.8
# Graph edges	82.2	82.6	78.8	78.1	90.8	56.1	78.1
# Business-Logic process	75.0	59.6	20.5	34.0	56.3	37.9	47.2

TABLE 8 Evaluation of the proposed for identifying business processes

Web applications metrics	WackoPicko	osCommerce	TomatoCart	VirtueMart	Openconf	Scarf	Average
Fitness	0.91	0.95	0.96	0.95	0.95	0.94	0.94
Simplicity	0.96	0.97	0.98	0.97	0.97	0.97	0.97
Precision	0.92	0.95	0.96	0.95	0.95	0.94	0.94

unknown. Furthermore, it was demonstrated that the precision of BLDAST in detecting the business processes of the selected applications was 94%, and that the generated user navigation graph was improved by 62.8% because of the detection of similar pages.

We intend to upgrade BLDAST to identify race conditions in web applications. Detecting race conditions in a web application depends highly on identifying its business logic. No business-aware approach exists for detecting race conditions in web applications. The approaches that have been proposed thus far for identifying race conditions lead to DoS attacks.

ORCID

Mitra Alidoosti https://orcid.org/0000-0003-0918-411X

REFERENCES

1. Verizon, *Data Breach Investigations Report*. New York, NY, USA, Tech. Rep., 2017, available at https://enterprise.verizon.com/content/dam/resources/reports/2017/2017_dbir.pdf (accessed 8 Nov. 2018).
2. The MITRE Corporation, *CVE: Common vulnerabilities and exposures*, McLean, VA, USA, available at http://www.cve.mitre.org (accessed 8 Nov. (2018).
3. Symantec, *Symantec Internet Security Threat Report*, Mountain View, CA, USA, Tech. Rep., Apr. 2016, available at https://www.symantec.com/security-center/threat-report (accessed 8 Nov. 2018).
4. *TrustwaveTrustwave Global Security Report*, Chicago, IL, USA, 2014, Tech. Rep., available at https://www.trustwave.com/Resources/Trustwave-Blog/The-2014-Trustwave-Global-Security-Report-Is-Here/ (accessed 8 Nov. 2018).
5. H. H. Jazi et al., *Detecting HTTP-based application layer DoS attacks on web servers in the presence of sampling*, Comput. Netw. **121** (2017), 25–36.
6. A. Wang et al., *Capturing DDoS attack dynamics behind the scenes*, in Proc. Int. Conf. DIMVA, Milan, Italy, July 9–10, 2015, pp. 205–215.
7. D. Holmes, *The F5DDoS protection reference architecture*, F5, Seattle, WA, USA, 2014, available at https://f5.com/Portals/1/Cache/Pdfs/2421/the-f5-ddos-protection-reference-architecture.pdf (accessed 8 November 2018).
8. OWASP, *Owasp testing guide* 2008, available at http://www.owasp.org/index.php/OWASP (accessed 8 November 2018).
9. White Hat Security, *Business logic assessments*, San Jose, CA, USA, 2016, available at https://www.whitehatsec.com/wp-content/uploads/2016/01/Sentinel_Dynamic_Business_Logic_Assessment_Solution_Brief.pdf (accessed 8 November 2018).
10. OWASP, *Testing for business logic*, available at https://www.owasp.org/index.php/Testing_for_business_logic (accessed 8 November 2018).
11. G. Pellegrino and D. Balzarotti, *Toward black-box detection of logic flaws in web applications*, in Proc. Int. Conf. NDSS Symp., San Diego, CA, USA, Feb.2014, pp. 23–26.
12. G. Deepa and P. S. Thilagam, *Securing web applications from injection and logic vulnerabilities: Approaches and challenges*, Inf. Softw. Technol. **74** (2016), 160–180.
13. W. Meng et al., *Rampart: Protecting web applications from CPU-exhaustion denial-of-service attacks*, in Proc. 27th USENIX Secur. Symp, Baltimore, MD, USA, Aug. 2018, pp. 393–410.
14. T. Petsios et al., *Slowfuzz: Automated domain-independent detection of algorithmic complexity vulnerabilities*, in Proc Int. Conf. Comput. Commun. Secur., Dallas, TX, USA, 2017, pp. 2155–2168.
15. J. Burnim, S. Juvekar and K. Sen, *WISE: Automated test generation for worst-case complexity*, in Proc. Int. Conf. Softw. Eng., Washington, DC, USA, May. 16–24, 2009, pp. 463–473.
16. O. Olivo, I. Dillig and C. Lin, *Detecting and exploiting second order denial-of-service vulnerabilities in web applications*, in Proc. Int. Conf. Comput. Commun. Secur., Denver, CO, USA, Oct. 12–16, 2015, pp. 616–628.
17. S. Son and V. Shmatikov, *SAFERPHP, Finding semantic vulnerabilities in PHP applications*, in Proc. Int. Conf. Programming, San Jose, CA, USA, June 2011, pp. 8:1–13.
18. J. Burnim et al., *Looper Lightweight detection of infinite loops at runtime*, in Proc. Int. Conf. Automated Softw. Eng., Washington, DC, USA, Nov. 16–20, 2009, pp. 161–169.
19. A. Gupta et al., *Proving non-termination*, in Proc. Int. Conf. Principles Programming Lang., San Francisco, CA, USA, Jan. 7–12, 2008, pp. 147–158.
20. R. Chang et al., *Inputs of coma: Static detection of denial-of-service vulnerabilities*, in Proc. IEEE Comput. Secur. Foundations Symp., New York, NY, USA, July 8–10, 2009, pp. 186–199.
21. M. Alidoosti and A. Nowroozi, *BLProM: Business-layer process miner of the web application*, in Proc. Int. Conf. Inf. Secur. Cryptol, Tehran, Iran, Aug, 2018, pp. 28–29.
22. X. Li and Y. Xue, *BLOCK: A black-box approach for detection of state violation attacks towards web applications*, in Proc. Int. Conf. Comput. Secur. Appl., Orlando, FL, USA, Dec. 5–9, 2011, pp. 247–256.
23. X. Li, W. Yan, and Y. Xue, *SENTINEL: Securing database from logic flaws in web applications*, in Proc. Int. Conf. Data Appl. Secur. Privacy, San Antonio, TX, USA, Feb. 7–9, 2012, pp. 25–36.
24. A. Doupé et al., *Enemy of the state: A state-aware black-box web vulnerability scanner*, in Proc. USENIX Secur. Symp., Bellevue, WA, USA, Aug. 8–10, 2012, pp. 523–538.
25. G. Pellegrino and D. Balzarotti, *Toward black-box detection of logic flaws in web applications*, in Proc. Netw. Distrib. Syst. Secur. Symp., San Diego, CA, USA, Feb. 2014, pp. 23–26.
26. G. Deepa et al., *DetLogic: A black-box approach for detecting logic vulnerabilities in web applications*, J. Netw. Comput. Appl. **109** (2018), 89–109.
27. F. Sun, L. Xu, and Z. Su, *Detecting logic vulnerabilities in e-commerce applications*, in Proc. Netw. Distrib. Syst. Secur. Symp., Los Angeles, CA, USA, 2014.
28. J. C. Buijs, B. F. van Dongen, and W. M. P. van der Aalst, *On the role of fitness, precision, generalization and simplicity in process discovery*, in Proc. Int. Conf. Move Meaningful Internet Syst., Heidelberg, Berlin, 2012, pp. 305–322.

AUTHOR BIOGRAPHIES

Mitra Alidoosti received the BS and MS degrees in computer engineering from the Department of Computer Engineering, Iran University of Science and Technology, Tehran, Iran, in 2009 and 2012, respectively. Currently, she is working toward the PhD degree in computer engineering at Malek-e-Ashtar University of Technology, Tehran, Iran. Her research interests are computer network security, VoIP and SIP security, and web-application security.

Alireza Nowroozi is a freelance consultant who advises government and private-sector-related industries on information technology. He has four-year experience as an academic staff and an IT post-doctoral position with Sharif University of Technology, Tehran, Iran. He is a specialist in artificial intelligence, cognitive science, software engineering, and IT security, and he is a co-founder of four IT startups.

Ahmad Nickabadi received the BS degree in computer engineering in 2004, and the MS and PhD degrees in artificial intelligence in 2006 and 2011, respectively, from Amirkabir University of Technology, Tehran, Iran. He is currently an assistant professor with the Department of Computer Engineering, Amirkabir University of Technology. His research interests include statistical machine learning and soft computing.