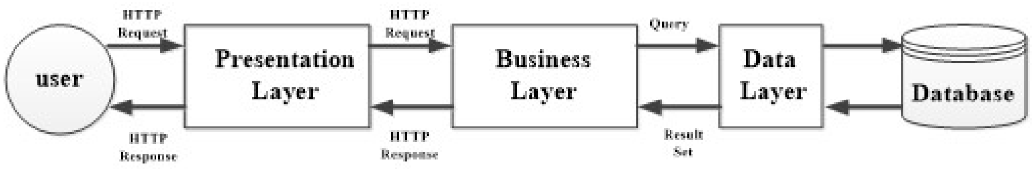


Figure 1. The Three-Layer Architecture of a Web Application [8].

flow. MiMoSA detects multi-step attacks by analyzing the relationship between the web application and the database, as well as the connections in the web application.

SENTINEL [11] and Pellegrino [3] use a *prevention approach* to identify business logic vulnerabilities in the form of a *black-box* approach.

SENTINEL [11] is a *black-box* approach for detecting logical weaknesses in database access. SENTINEL generates a state machine of the web and extracts a set of invariants from observed SQL queries and responses and session variables as the application specification. Any SQL query that violates defined invariants is identified as an attack.

Pellegrino et. al. [3] propose a *black-box* technique to detect logic vulnerabilities in web applications. This technique extracts behavioral patterns from network traces in which the user interacts with a certain application's functionality. First, the web application is modeled and then attack vectors are applied to the model.

Our previous works, BLDAST [12, 13] and BLTOCTTOU [14] use a prevention approach to identify business logic vulnerabilities in the form of a black-box approach. BLProM [15] can be used as an input for BLTOCTTOU and BLDAST.

BLDAST [12, 13] is a dynamic and a black-box vulnerability analysis approach that identifies business logic vulnerabilities of a web application against flooding DoS attacks. BLDAST assesses web application resiliency against flooding DoS attacks. It can take into account the business processes of a web application. BLDAST selects critical pages in business processes. A critical page has considerable response time. Therefore a critical process can enforce heavy load into the target and lead the web server to become unresponsive. The goal of the BLDAST is to find these critical processes within the web applications.

BLTOCTTOU [14] is a black-box dynamic application security tester for detecting business logic vulnerabilities against race condition attacks. BLTOCTTOU identifies vulnerabilities with the help of finding the business processes of the web application. BLTOCTTOU detects business processes that interact with each other; one process should set the value of a variable and the other should read or write that variable. To identify the race condition, BLTOCTTOU first executes identified processes sequentially and then executes them in reverse order. At last, it evaluates the outputs of these two modes. If they are different, the web application is vulnerable to a race condition.

2.2 Clustering Web Pages

Crescenzi [16] presented an approach to cluster web pages based on the page structure. The structural similarity between web pages is defined by DOM trees of their hyperlinks. The final clusters are used to build a model that describes the structure of the site according to classes of pages and their connectivity.

3 BUSINESS-LAYER PROCESS MINER

In this paper, the BLProM is proposed to identify business processes of the web application and we use its outputs as the input in the web application security testing in the business layer. Then by analyzing the interaction between business processes, business layer vulnerabilities can be detected.

BLProM first preprocesses normal user HTTP traffic. Then extracts web application pages in the traffic. BLProM clusters similar pages to prevent the infinite growth of the user navigation graph. Detected clusters are graph nodes and the graph edges are the relations between detected clusters. Based on detected nodes and edges, the user navigation graph is extracted. Then BLProM extracts business processes from the navigation graph. BLProM has two main steps:

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

Figure 2 shows the proposed steps to identify the business process in the web application. In the following, we will explain each of these steps in detail.

3.1 Extracting the User Navigation Graph

First, the normal user starts to crawl the web application. The traffic of a normal user is captured and stored. It should be noted that the user permission

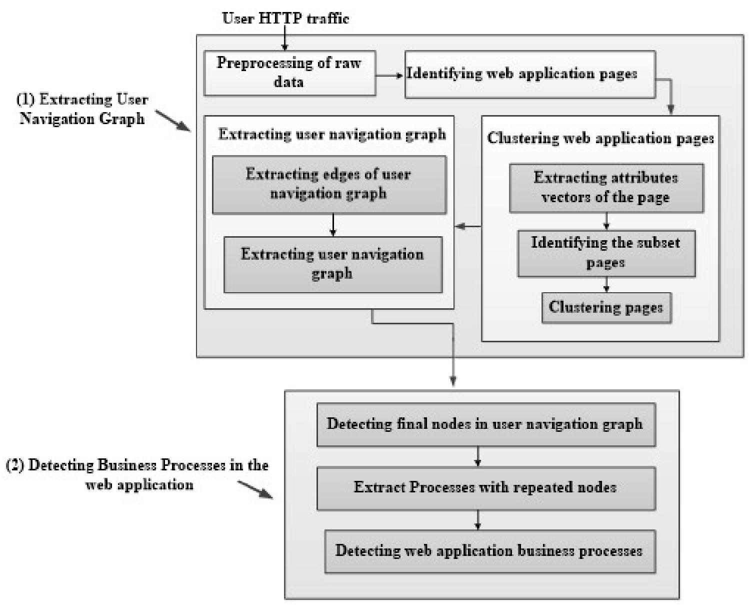

User HTTP traffic
(1) Extracting User Navigation Graph
Preprocessing of raw data
Identifying web application pages
Extracting user navigation graph
Extracting edges of user navigation graph
Extracting user navigation graph
Clustering web application pages
Extracting attributes vectors of the page
Identifying the subset pages
Clustering pages
(2) Detecting Business Processes in the web application
Detecting final nodes in user navigation graph
Extract Processes with repeated nodes
Detecting web application business processes

Figure 2. Black-Box Approach to Detect Web Application Business Process.

level can be different according to the role of the user and consequently the user navigation graph varies depending on the permission level. In this paper, the normal user has the user-level permission and searches through related permissible pages. The normal user crawls all different parts of the application that are allowed to search.

BLProM initially extracts the user navigation graph from the stored traffic; this is performed through steps as follows:

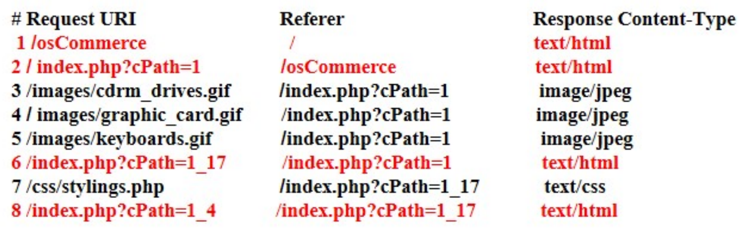
1. Preprocessing of raw input data
2. Identifying existing web application pages in the stored traffic
3. Clustering the web application pages
4. Extracting the user navigation graph

1) *Preprocessing of raw input data*
In the preprocessing step, the BLProM cleans the data and removes irrelevant data samples. In this paper, only HTTP requests and responses are used. For the responses, only those with successful status codes 200 and 209 are employed. The BLProM removes responses with failure status codes as well as their corresponding requests. Additionally, in this paper, only GET and POST requests are needed and the remaining ones are discarded.
2) *Identifying the web application pages in the stored traffic*

2) *Identifying the web application pages in the stored traffic*
Each page of the application can be represented as a pair (main request, corresponding response to the main request). To identify the web application pages, it is first necessary to detect the main requests in the traffic. After identifying the main requests, the corresponding responses must also be identified.

Identifying the main HTTP requests in the traffic: In the user's stored traffic, there are both the main HTTP requests that lead to loading the web application pages and the secondary HTTP requests that are responsible to load a file, image, etc. of the page. The BLProM must distinguish between the main and secondary requests. In other words, when the main requests are identified, the remaining ones are considered as secondary requests. In this way, any request that its Referer header is different from the Referer of the previous request is considered as the secondary requests and the previous one is the main request. Additionally, the first request in the traffic is considered the main request because the first request does not include the Referer. It should be noted that the Referer header field in the HTTP request indicates the URL of the previous page the user visited.

Identifying the corresponding responses to the main requests: To identify the corresponding responses to the main requests, it is only needed to select the responses that their content-type field is text/html. Because the corresponding response to the secondary requests is often a file, photo, etc. while the corresponding response to the main requests is in the form of text and HTML. Such responses are the main responses in the HTTP traffic. After identifying the main requests and responses in the traffic, each pair (main request, corresponding responses to the main request) indicates a page of the web application. It



#	Request URI	Referer	Response Content-Type
1	/osCommerce	/	text/html
2	/ index.php?cPath=1	/osCommerce	text/html
3	/images/cdrm_drives.gif	/index.php?cPath=1	image/jpeg
4	/ images/graphic_card.gif	/index.php?cPath=1	image/jpeg
5	/images/keyboards.gif	/index.php?cPath=1	image/jpeg
6	/index.php?cPath=1_17	/index.php?cPath=1	text/html
7	/css/stylings.php	/index.php?cPath=1_17	text/css
8	/index.php?cPath=1_4	/index.php?cPath=1_17	text/html

Figure 3. Number of Consecutive Requests.

should be noted that in responses of the main requests, all secondary requests exist that lead to load the web page.

Identifying whether the last request in the traffic is the main request or not: If the last response in the traffic is the main response, the last HTTP request is the main request as well.

For example, in Figure 3, requests 1, 2, 6, and 8 are the main requests that are shown in red.

The pseudocode of the algorithm used by the BLProM to identify web application pages is shown in Algorithm 1. As mentioned before, each web application page can be indicated by a pair (main request, corresponding responses to the main request). The pseudocode in Algorithm 1 can be divided into four main parts:

1. Extracting main requests in the traffic (line 13-26)
2. Extracting corresponding responses to the main requests (line 27-33)
3. Identifying whether the last request in the traffic is the main request or not (line 34-37)
4. Extracting web application pages (lines 38-43)

In line 9, first, all requests in the traffic either the main requests or secondary ones are extracted and put in the *HTTPRequest* variable. In line 10, existing responses in the traffic are added to the *HTTPResponse* variable. In lines 11 and 12, the total number of requests and responses is calculated. In line 15, it is checked whether the current request is the first request, if yes, it is considered as the main request and included in the *MainRequest* variable. In line 21, it is checked whether the *Referer* field of the current request is different from the *Referer* field of the previous request, if yes, the previous request is considered as the main request.

In line 29, it is checked whether the content-type field of the current response is text/html. If the given condition is met, the current request is considered as the main request.

In line 35, it is checked whether the last HTTP response is the main response, if yes, the last request is considered as the main request as well.

In line 41, all main requests and responses are the web application pages which are shown as pairs (request, response).

3) *Clustering the web application pages*

In this step, BLProM clusters the web application pages. Clustering aims to put similar pages in the same cluster. This is helpful to prevent the infinite growth of the user navigation graph. The pages in the same cluster are similar to each other.

At this stage, all pages in the user's stored traffic have been extracted as pairs (main request, corresponding response to the main request). Each pair (main request, corresponding response to the main request) shows one page of the application. To extract the optimal user navigation graph, the similar extracted pages must be identified and clustered. In the user navigation graph, nodes indicate the application's unique pages and edges represent the link between the pages. To identify similar pages, a criterion should be considered by the purpose of the clustering. The type of operations that the user can perform on the page is considered as a measure for the separation of pages. In other words, two pages are similar if the user can perform the same operations on them. For example, consider two pages such that both contain only a button but the title of the buttons is different where in the first page the title is "continue" and for the second one, it is "save". These two pages are different because the user performs different operations on them. Thus, according to the criteria specified for the similar pages, the following pages are considered similar in web applications:

- In the online shop application, if the pages related to the shopping cart of goods even if they contain different items, they are considered similar pages.
- In the online shop application, the profile pages of each product are similar. because they have the same HTML structure in terms of the important HTML elements.
- Pages that display search results with different keywords are similar.
- If the structure of pages is a subset of another page in terms of the important HTML elements, these pages are considered similar.
- Two pages that both contain user comments are considered similar even if the contents are about different products.

Algorithm 1 The Pseudocode for Extracting Web Application Pages From the Traffic

INPUT: HITRtraffic:$\{HttpMessage_1, HttpMessage_2, HttpMessages_3, \dots, HithMessage_n\}$
OUTPUT: WebPages as a set of $(Request_i, Response_i)$

```
1: Begin
2: let JWebPages = {}; // set of web pages
3: let HttpRequest= {}; I/set of HTTP Requests
4: Iet HittpResponse = {}; // set of HTTP Responses
5: let MainRequest= {}; // set of main HTTP Requests
6: let MainResponse = {}; // set of main HTTP Responses
7: let i, k= 1; // counter for current HttpMessage
8: let LastReferer = ∅ , NewRefere = ∅
9: HttpRequest=ExtractReg(HITRtraffic);//extract HTTP Requests from HTTP traffic
10: HttpRespense - ExtractResp(HITRtraffic);//extract HTTP Response from HTTP traffic
11: n = extractNumber(HttpRequest) //extract total number of HTTP Requests
12: m = extractNumber(HttpResponse) //extract total number of HTTP Responses
13: // extract Main HTTP Request from HTTPRequests
14: for i: 1 . . . n do
15:     if ($i = 1$) then
16:         add MainRequest ← $HttpRequest_i$ // First Request is a Main Request
17:         $LastReferer$ ← Referer of $HttpRequest_i$;
18:     else
19:         $NewRefere$ ← Referer of $HttpRequest_1$;
20:     end if
21:     if ($NewReferer \neq LastReferer$) then
22:         add MainRequest ← $HttpRequest_{i-1}$;
23:         $LastReferer \leftarrow NewRefere$;
24:     end if
25: end for
26: //extract Main HTTP Response from HTTPResponse
27: for k : 1 . . . m do
28:     if (content-type of $HTTPResponse_k$ = text/html) then
29:         $addMainRespense \leftarrow HttpRespense_k$
30:     end if
31: end for
32: // checking last request is main request or not
33: if ($HTTPResponse_m isinMainResponse$) then
34:     $addMainRequest \leftarrow HTTPRequest_m$
35: end if
36: //extract set of web pages
37: size = extractNumber(MainRequest) //extract total number of Main HTTP Requests
38: for j: 1 . . . size do
39:     $WebPages \leftarrow (MainRequest_j)MainResponse_j)$;
40: end for
41: return WebRages;
42: end
```

Table 1. Attribute Vector of a Page in osComerce Web Application.

inputs	null
Buttons	html.body.button.Reviews# html.body.div.div.div.div.form.div.div.span.s pan.button.Add to Cart
anchors	null
image	html.body.div.div.div.a.img

```
<html>
          <title> Hello! </title>
          <body>
             <p>
                <button value= "continue"> Continue</button>
             </p>
          </body>
</html>
```

Figure 4. An Example of HTML Code.

Definition of Document Object Model (DOM) path for an HTML element: DOM path of an element is the position of the element in the HTML code.

For example, in Figure 4 the DOM path of the button (DOM_{button}) is Document.Html.Body.P.Button.

Definition 1. [Similar Pages] the similar pages are those the user can perform the same operations on them and are identical in terms of the position of the important HTML elements in the page. The important HTML elements in the page include buttons, images, inputs, and anchors.

The clustering process includes three steps:

1. Extracting attributes vectors of the page
2. Identifying the subset pages
3. Clustering pages

In the following, these steps are discussed in detail.

1. **Extracting attributes vectors of the page**
   The BLProM shows each page of the application as a pair (main request, response). In this step, BLProM extracts the corresponding attributes vectors of each page by applying a data mining operation on the above pair. The BLProM models each page using the following attribute vector:

   WebPages = the total pages in an application
   $\forall$ w ϵ WebPages
   w= $(DOM_{inputs},\ DOM_{buttons},\ DOM_{anchors},\ DOM_{imgs})$
   $DOM_{inputs} = \prod_i^n DOM(input_i)$
   $DOM_{buttons} = \prod_i^n DOM(buttons_i)$
   $DOM_{anchors} = \prod_i^n DOM(anchor_i)$
   $DOM_{imgs} = \prod_i^n DOM(img_i)$

   - DOM(input): DOM path of $< input >$ tag in the page + the value of type attribute in the $< input >$ tag + the value of name attribute in the $< input >$ tag (in the absence of name attribute, the value attribute is considered).
   - DOM(button): DOM path of the button in the page + the title of button.
   - DOM(anchor): DOM path of $< a >$ tag in the page.
   - DOM(img): DOM path of the existing image in the page.

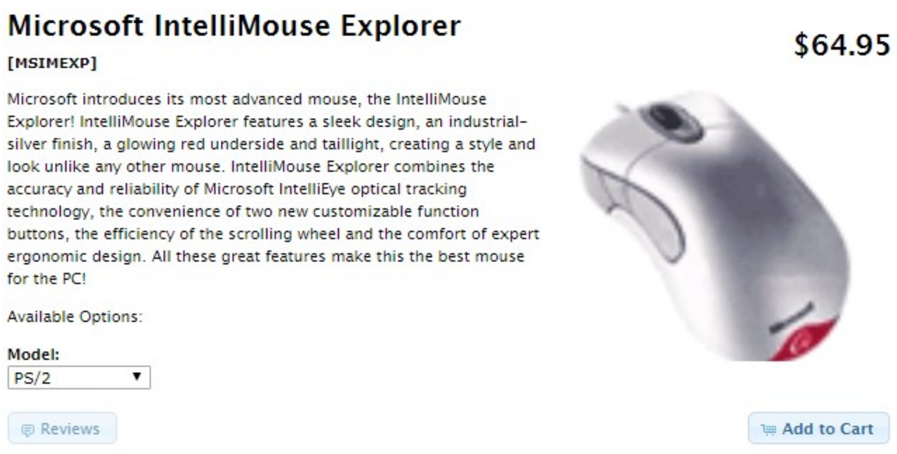
   Suppose the web application page contains several buttons; in this case, the second element of the page attribute vector is a set of the DOM paths of buttons in the page that are separated by "#". Figure 5 shows one of the osCommerce [1] web application pages. Table 1 shows the attribute vector of the page in Figure 5. As shown, the input element and the anchor elements are null, it means the page does not contain the above tags.

2. **Identifying similar pages**
   After extracting the attribute vector of each page, it is necessary to identify similar pages. Those pages that their attribute vectors are a subset of another page or have fully similar attribute vectors are considered as similar pages.

   According to Definition 1, the attribute vector of each web application page has four elements. The attribute vector of page 1 is considered the same as the attribute vector of page 2 if:

   - All vector elements of page 1 equal to corresponding elements in the vector of page 2.
   - All vector elements of page 1 are a subset of corresponding elements in the vector of page 2.
   - All vector elements of page 2 are a subset of corresponding elements in the vector of page 1.

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



Microsoft IntelliMouse Explorer

[MSIMEXP]

$64.95

Microsoft introduces its most advanced mouse, the IntelliMouse Explorer! IntelliMouse Explorer features a sleek design, an industrial-silver finish, a glowing red underside and taillight, creating a style and look unlike any other mouse. IntelliMouse Explorer combines the accuracy and reliability of Microsoft IntelliEye optical tracking technology, the convenience of two new customizable function buttons, the efficiency of the scrolling wheel and the comfort of expert ergonomic design. All these great features make this the best mouse for the PC!

Available Options:

Model: PS/2

Figure 5. A Page of osCommerce Application.

Algorithm 2 The Pseudocode for Identifying the Similar Pages

```
INPUT: $w_1 = (DoM_{w1}(input), DoM_{w1}(button), DoM_{w1}(anchor), DoM_{w1}(img))$,
$w_2 = (DoM_{w2}(input), DoM_{w2}(button), DoM_{w2}(anchor), DoM_{w2}(img)$
OUTPUT: Boolean flag // true means two pages are the same
1: Begin
2: Let flag, input, button, anchor, img=false;
3: if $DoM_{w1}(input) \subseteq DoM_{w2}$ (input) or $DoM_{w2}(input) \subseteq DoM_{w1}(input)$ then
4:     input=true;
5: end if
6: if $DoM_{w1}(button) \subseteq DoM_{w2}$ (button) or $DoM_{w2}(button) \subseteq DoM_{w1}(button)$ then
7:     button=true;
8: end if
9: if $DoM_{w1}(anchor) \subseteq DoM_{w2}$ (anchor) or $DoM_{w2}(anchor) \subseteq DoM_{w1}(anchor)$ then
10:    anchor=true;
11: end if
12: if $DoM_{w1}(img) \subseteq DoM_{w2}(img)$ or $DoM_{w2}(img) \subseteq DoM_{w1}(img)$ then
13:    img=true;
14: end if
15: if input and button and anchor and img then
16:    flag=true;
17: end if
18: return flag;
19: end
```

- If one or more vector elements of page 1 are a subset of their corresponding elements in the vector of page 2, the rest of the vector elements of page 1 must be the same with their corresponding elements in the vector of page 2.
- The null element is a subset of every element.

Similar pages are identified according to the above-mentioned attributes. Algorithm 2 illustrates the pseudocode for identifying similar pages.

3. **Clustering web application pages**
After identifying the similar pages, they are put in the same cluster. The pages in a cluster are similar to each other and refer to a unique page of the application. Algorithm 3 shows the pseudocode for clustering web pages. In line 7, it is checked whether two pages wi and wj are the same, if yes, they are put in the same cluster.

4) *Extracting user navigation graph*
In this step, BLProM connects the obtained clusters that each one represents a unique web application page. Each cluster has a set of similar pages, each of these pages has URI and Referer field. Thus, each cluster contains a set of URIs and a set of Referers for the pages in that cluster. It should be noted that the Referer field is the URI

Algorithm 3 The Pseudocode for Clustering the Web Application Pages

INPUT: WebPages = $\{w_1, w_2, w_3, \ldots, w_n\}$
OUTPUT: the web application model M as a set of web page clusters C;
the web page clusters C as a set of web pages;

```
 1: Begin
 2: Let $M = \varnothing$; // set of page clusters
 3: Let k=1; // number of web page clusters
 4: for $i : 1 \ldots n$ do
 5:     $C_k \leftarrow w_i$
 6:     for $j = i + 1 \ldots n$ do
 7:         if $SimilarWebPages(w_i, w_j)$ then
 8:             WebPages $\leftarrow WebPages - w_i$;
 9:             n= length of WebPages;
10:             $C_k \leftarrow w_j$
11:         end if
12:     end for
13:     k + +;
14: end for
15: return C;
16: end
```

of the previous web application page that the user visited. For extracting the edges of user navigation graph, the produced clusters are checked to find which Referer set of clusters has the intersection with the URI set of the cluster. When the cluster is found, these two clusters are connected. Suppose that the URI set of cluster C1 has an intersection with the Referer set of cluster C2, then the path from cluster C1 to cluster C2 (C1 $\rightarrow$ C2) is created. In other words, the edge C1C2 is one of the edges in the user navigation graph. Algorithm 4 shows the pseudocode for extracting the edges from the user navigation graph. The URI set and the Referer set of each cluster are respectively obtained according to lines 5 and 6 of the pseudocode in Algorithm 3. In line 10, if the intersection of the URI set of each cluster with other clusters' Referer set is not null, the CiCj edge is added to the edge set of the graph.

In this step, BLProM connects the obtained clusters that each one represents a unique web application page. Each cluster actually has a set of similar pages, each of these pages has URI and Referer field. Thus, each cluster contains a set of URIs and a set of Referers for pages in the cluster. It should be noted that the Referer field is actually the URI of the previous web application page that the user visited. For extracting the edges of user navigation graph, the produced clusters are checked to find which Referer sets of clusters has the intersection with the URI set of the cluster, when the cluster is found, these two clusters are connected. Suppose that the URI set of cluster C1 has intersection with the Referer set of cluster C2, then the path from cluster C1 to the cluster C2 (C1 $\rightarrow$ C2) is created. In other words, the edge C1C2 is one of the edges in the user navigation graph. Algorithm 4 shows the pseudocode for extracting the edges from the user navigation graph. The URI set and the Referer set of each cluster are respectively obtained according to the lines 5 and 6 of the pseudocode in Algorithm 3. In line 10, if the intersection of URI set of each cluster with other clusters' Referer set is not null, CiCj edge is added to the edge set of the graph. The user navigation graph is created according to the algorithm in Algorithm 5, where nodes are the clusters set and the identified edges are the path between clusters. In the created graph, each node indicates the unique page and the edges show the path between pages.

Definition 2. [The User Navigation Graph] This graph is shown with tuple<C0, C, E> where C is the set of nodes in the graph, C0 $\in$ C is the first (initial) node in the graph and E $\subseteq$ C $\times$ C is the set of edges in the graph.

Algorithm 5 shows the pseudocode for extracting the user navigation graph. Line 7 indicates a cluster that contains the initial page of the application. It is considered as the initial node of the graph.

3.2 Identifying Business Processes in the Application

To identify the web application business processes in the user navigation graph, it is necessary to define the process and final node and then define the business process.

Definition 3. [The Application Process (P)] The

Algorithm 4 The Pseudocode for Extracting Edges From the User Navigation Graph

INPUT: C = $\{C_1, C_2, C_3, \ldots, C_k\}$ //web pages clusters as graph nodes
WebPages= $\{w_1, w_2, w_3, \ldots, w_n\}$ //set of web pages
OUTPUT: the web application graph edges E as a set of edges

```
1:  Begin
2:  Let E = ∅; // set of web application graph Edges
3:  for $j : 1 \ldots k$ do
4:      Let $URI_{cj}, Referer_{cj} = \varnothing$;
5:      $URI_{cj} = URI_{cj} \cup$ ExtractURI (w) for any w $\in C_j$
6:      $Referer_{ck} = Referer_{ck} \cup$ ExtractReferer(w) for any w $\in C_j$
7:  end for
8:  for i: $\ldots k$ do
9:      for $j : i + 1 \ldots k$ do
10:         if $(URI_{ci} \cap Referer_{cj} \neq null)$ then
11:             $E \leftarrow E + C_iC_j$
12:         end if
13:     end for
14: end for
15: return E;
16: end
```

Algorithm 5 The Pseudocode for Extracting the User Navigation Graph

INPUT: C= $\{C_1, C_2, C_3, \ldots, C_k\}$ //web pages clusters as graph nodes
w_1 // First web page
OUTPUT: the web application navigation graph $< C_0, C, E >$

```
1:  Begin
2:  Let $C_0 = \varnothing$
3:  E=ExtractGraphEdges; //set of web application graph Edges
4:  for $j : 1 \ldots k$ do
5:      if $C_k$ contains $w_1$ then
6:          $C_0 = C_o + C_k$
7:      end if
8:  end for
9:  return $C_0, C, E$;
10: end
```

process P in the application is a sequence of nodes and edges in the user navigation graph like $E_1, E_2, \ldots, E_k, whereE_i \in E, andE_i = C_{i-1}C_i$.

Algorithm 6 shows the pseudocode for extracting processes.

Definition 4. [the final nodes in the user navigation graph (F)] The final nodes refer to the completion of a business process that occurs when the application reaches there.

The final nodes can be detected by examining the HTTP responses. For example, in the process of buying a product, a phrase like ”Thank you for your purchase” is displayed after completion of the purchase. By specifying a set of these phrases and searching them in the responses, the final nodes can be identified. Some keywords used for identifying the final node are Thank, Congratulations, Successfully, Log Off and Search Results. Additionally, some buttons in the web application page are good signs that help the identification of the final node in the graph. The examples of final nodes include the page after clicking the “save” button, the page after clicking the “creation” button and the page after clicking the “submit” button.

Definition 5. [the application business process] The business process BP in the application is a process that has at least one of the following conditions:

1. The first node of the process is the initial node in the user navigation graph (C_0) and the process end node is the final node in the user navigation graph (F).
2. If the process passes its first node again, it means

Algorithm 6 The Pseudocode for Extracting Processes

INPUT: the web application first node C_0
the web application Graph edges E
OUTPUT: the web application graph process P as a set of web application process

```
1: Begin
2: Let $P_0 = \varnothing$; // set of web application processes
3: Let StartEdge= $\varnothing$; // set of graph edge that begin from $C_0$
4: StartEdge= ExtractGraphEdges($C_0$) //Extract all edges from $C_0$
5: EndPoint = ExtractEndPoint(StartEdge) //Extract the end point of edges
6: if ($ExtractProcess(EndPoint, E) \neq null$) then
7:     return P=E+ExeractProcess(Endpoint, E); for any edge E $\in$ StartEdge
8: else
9:     return E;
10: end if
11: end
```

the first node and the end node of the process are the same and the process length is greater than two. (If there is a return to the passed node in the process and the created loop length is more than two, this is a business process).

All processes in the graph from the initial node (the application initial page) to the identified final nodes, as well as the processes of their initial node and the final node are the same; and all of them are the application business processes. Algorithm 7 shows the pseudocode for identifying the application business process. In line 4, the application business processes are extracted. In line 5, the final nodes of the graph are extracted. In line 7, the processes that start with the initial node and end with the final node are identified as the business process are stored in the variable BP. In line 9, the processes with repeated nodes are detected and in line 11, among the detected processes, if their initial node and their final node are the same and their length is greater than two, they are added to the variable BP as the business process.

4 EXPERIMENTAL RESULTS

The testbed used in this section is a network consisting of a web server (test target) and two clients (BLProM system and legal user). The web server and the clients are loaded on a virtual machine. The web server and the clients' profiles are shown in Table 2.

The web applications listed in Table 3 are installed on the web server (test target) and then we plan to identify the business layer of the web applications.

The legal user first starts using the selected web applications. The user crawls all permitted parts of the web application. HTTP traffic of the legal user is given to BLProM as its input.

5 EVALUATION

BLProM's goal is to identify the business layer of the web application. We can identify business logic vulnerability by identifying the business layer of the web application. BLProM detects the business processes of the web application. Identifying business processes is the main step in dynamic security testing of the web application in the business layer.

We compare BLProM with OWASP ZAP. The OWASP Zed Attack Proxy (ZAP) is a free web scanner. It scans web applications and automatically finds some security vulnerabilities. ZAP is the only free web scanner that has API for extracting web application graph. The web application pages are graph nodes and the relations among pages are shown as graph edges. The main difference between BLPRoM and ZAP is in detecting similar pages. ZAP cannot detect similar pages in the web application but BLPRoM can. ZAP's graph only shows the relation among scanned pages but BLProM generates the optimal graph. About the accuracy of the generated graph, both BLProM and ZAP are the same.

To evaluate the proposed approach, we first show the accuracy of clustering by the following criteria:

- True Positive: Samples that fit well into their correct clusters.
- False Positive: Samples that fit in a cluster that do not belong to that cluster.
- False Negative: Samples that do not fit in a cluster but they belong to that cluster.
- Recall: It is calculated by the following formula:

$$recall = \frac{TruePositive}{TruePositive+FalseNegative}$$

- Precision: It is calculated by the following formula: