2018 15th International ISC (Iranian Society of Cryptology) Conference on Information Security and Cryptology (ISCISC)
August 28-29, 2018; Shahid Rajaee Teacher Training University - Tehran, Iran

BLProM: Business-layer Process Miner of the web application

Mitra Alidoosti
Information, Communication and Security Technology
Malek-e-Ashtar University of technology
Tehran, Iran
Alidoosti@mut.ac.ir

Alireza Nowroozi
Computer Engineering
Sharif University of Technology
Tehran, Iran
Nowroozi@ce.sharif.edu

***Abstract*— Web application vulnerability scanners cannot detect business logic vulnerabilities (vulnerabilities related to logic) because they are not able to understand business logic of the web application. In order to identify business logic of the web application, this paper presents BLProM, the black box approach that identifies business processes of the web application. Detecting business processes of the web applications can be used in dynamic security testing to determine business logic vulnerabilities in the web applications. BLProM first extracts navigation graph of the web application then identifies business processes from the navigation graph. The evaluation conducted on three well-known open source web applications shows that BLProM is able to detect business logic processes. Experimental results show that BLProM improves web application scanning because it clusters web application pages and prevent scanning similar pages. The proposed approach is compared to OWASP ZAP, an open source web scanner. We show that BLProM improves web application scanning about %96.**

Keywords*— Business layer, business process, navigation graph

I. INTRODUCTION

Most of vulnerabilities reported in CVE [1] database are related to the web application vulnerabilities. The number of security breaches increased by 35.5% in 2015 compared to last year. Most of these attacks were applied to military, e-commerce and medical applications [2]. Business logic vulnerabilities affect web applications security as the most potent vulnerabilities. So far, there are subtle vulnerabilities related to the web application logic that are still discovered manually. Automated scanners cannot detect business logic flaws in applications because scanners are not able to understand the context [3]. Such vulnerabilities can only be detected through manual testing and be relied on tester creativity and skills [4].

There is no formal definition for business logic vulnerabilities. It is very difficult to detect business logic vulnerabilities and this type of vulnerability cause serious damage in case of misuse. Understanding context is difficult for automated tools, so penetration testers are responsible for detecting business logic vulnerabilities. Since business logic vulnerabilities are application-specific, it's hard to detect these types of vulnerabilities [3].

Web applications do not have any formal documentation describing their internal states and expected user behavior. The lack of such a document makes it difficult to detect business logic vulnerabilities. For example, adding a specific item several times in a shopping cart is a common feature but repeated usage of discount code is a kind of business logic vulnerabilities. A person easily understands the difference between these two scenarios, while a scanner without an appropriate model cannot distinguish between them [4]. Researches [5-7] have been conducted to automatically detect business logic vulnerabilities but they are used for small applications. In addition, application source code is required to generate the appropriate model of the application [4]. So nowadays; an automatic tool which can finds business logic vulnerabilities is required.

In this paper, we propose BLProM, a black-box technique for identifying web application business layer. BLProM by identifying business processes in the web application provides the ability to identify business layer vulnerabilities. In other words, in order to dynamic security testing of web application in the business layer, it is necessary first to identify business processes of the web application (detecting business logic of the web applications). Then, by analyzing the processes, the business layer vulnerabilities are identified. The BLProM output is the Web application business processes that are used as input in dynamic security testing in the business layer. The proposed approach is independent of the technology used in the web applications and automatically finds business processes. In addition, we show that BLProM improves web application scanning, because it detects similar pages in the web application, and prevents scanning of similar pages. Comparing the results of the web application scanning, by BLProM and OWASP ZAP (web application open source scanner), shows that BLProM improves web application scanning about 96%. In summary, this paper makes the following contributions:

- We present BLProM, a black-box technique for detecting web application business-layer.
- We present a new black-box approach for clustering web pages
- We show that web application scanning improved about %96 by clustering web pages.

978-1-5386-7582-3/18/$31.00 ©2018 IEEE

II. Background and Related Work

The business layer determines business logic of web applications. The business layer is responsible for data processing and data management and specifies business logic policies and rules. In addition, this layer validates the input data. Fig. 1 shows three layer architecture of a web application and the position of the business layer in the web application.

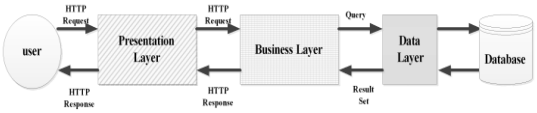

Fig. 1. Three layer architecture of a web application [8]

The presentation layer Interacts with the user and receives user input data. The business layer processes the business rules and submits the received data from the presentation layer to the third layer. The data access layer interacts with the database.

After receiving data from user, the data is available to the business layer. The Web application uses the data to run business processes. Every business process has several steps that must be implemented respectively and processes may interact with each other.

There are two approaches to prevent business logic attacks: 1) identifying attacks at runtime (*defense approach*) and 2) identifying logic vulnerabilities in the *web* applications (*prevention approach*). In the *defense approach*, an attack is reported whenever the behavior of the web application deviates from the normal behavior. In the *prevention approach*, attack vectors are used to identify business logic vulnerabilities.

BLOCK [9] and Swaddler [10] use *defense approach* to prevent business logic attacks. BLOCK First obtains the behavioral model of the web application by observing the interaction of the clients and the web application. It extracts a set of constants from the request / response sequence and session variables. BLOCK Identifies any request or response that violates identified constants as an attack.

Swaddler provides an anomaly detection method for detecting attacks. In fact, it uses anomaly in the internal state of the web application to detect vulnerabilities. In other words, the web application's internal state is monitored in the learning phase and the normal values of the web application state are extracted which define the web application profile. Then in the detection phase, abnormal states are identified.

MiMoSA [5] uses *prevention approach* in the form of *white-box* approach to identify business logic vulnerabilities. MiMoSA first provides a web application model based on the web application's state and workflow. 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.

BLDAST [11] and BLTOCTTOU [12] use *prevention approach* to identify business logic vulnerabilities in the form of *black-box* approach.

BLDAST [11] is a dynamic and 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 is able to take into account the business process of a web application.

BLTOCTTOU [12] is a black-box dynamic application security tester in order to detecting business logic vulnerabilities against race condition attacks. BLTOCTTOU identifies vulnerabilities with help of finding business processes of the web application.

BLProM can be used as an input for BLTOCTTOU and BLDAST.

III. 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 input in the web application security testing in the business layer. Then by analyzing 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 infinite growth of 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.

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

A. Extracting the user navigation graph

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

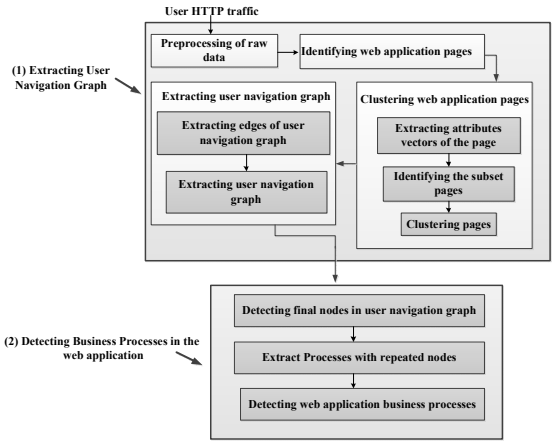

Fig. 2. Black-box approach to detect web application business process

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

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 fact, when the main requests are identified, the remaining ones are considered as the secondary requests. In this way, any request that its Referer header is different from the Referer of 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 filed in the HTTP request indicates the URL of previous page the user visited.

Identifying the corresponding responses to the main requests: in order 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 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 a 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.

3) Clustering the web application pages

In this step, BLPROM clusters the web application pages. The aim of clustering is to put similar pages in a same cluster. This is helpful to prevent the infinite growth of the user navigation graph. The pages in a 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. In order 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.

In order to identify similar pages, a criterion should be considered in accordance with the purpose of the clustering. The type of operations that the user is able to 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 so 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.

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 details.

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 $\in$ WebPages w= (DOM_{inputs}, $DOM_{buttons}$, $DOM_{anchors}$, DOM_{imgs})

$DOM_{inputs}= \Pi_i^n DOM(input_i)$

$DOM_{buttons}=\Pi_i^n DOM (buttons_i)$

$DOM_{anchors}= \Pi_i^n DOM (anchor_i)$

$DOM_{imgs}= \Pi_i^n DOM (img_i)$

- 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 "#".

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 a fully similar attribute vectors are considered as 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.

4) Extracting user navigation 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 C_1 has intersection with the Referer set of cluster C_2, then the path from cluster C_1 to the cluster C_2 ($C_1 \rightarrow C_2$) is created. In other words, the edge C_1C_2 is one of the edges in the user navigation graph.

Now the user navigation graph can be created according to Definition 2, 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 $<C_0, C, E>$ where C is the set of nodes in the graph, $C_0 \in C$ is the first (initial) node in the graph and $E \subseteq C \times C$ is the set of edges in the graph.

B. Identifying business processes in the application

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

Definition 3- the application process (P): The process P in the application is a sequence of nodes and edges in the user navigation graph like $E_1, E_2, \ldots, E_k$, where $E_i \in E$, $E_i=C_{i-1}C_i$.

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 the 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, 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 click the "save" button, the page after click the "creation" button, the page after click 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 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 their initial node and the final node are the same are the application business processes.

IV. EXPERIMENTAL RESULTS AND EVALUATION

The test bed used in this section is a network consisting of a web server (test target) and one client (legal user). The web server and the client are loaded on a virtual machine. The web server and the client's profiles are shown in Table 1.
The web applications listed in
Table 2 are installed on the web server (test target) and 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 as input to BLProM.

Table 1. Test bed profiles

Web server (test target)	CPU: Pentium dual core-2.20 GHZ OS: windows 8.1 VMware CPU: 1GHZ VMware RAM: 1G VMware OS: windows 7
Client (legal user)	CPU: Pentium dual core i3-3210 GHZ OS: windows 7 VMware CPU: 1GHZ VMware RAM: 1G VMware OS: windows 7

Table 2. Selected web applications for evaluation

Web application	Description
TomatoCart-1.1.8.6.1	e-commerce
osCommerce-2.3.4	e-commerce
WackoPicko	Web application for Sharing picture

Table 3:Evaluation of the clustering of selected web application pages

Web application / Criteria	WackoPicko	Tomatocart	osCommerce
#samples	89	150	210
#clusters	29	66	40
true positive	65	146	205
false positive	24	4	5
false negative	23	3	4
recall	0.74	0.98	0.98
precision	0.73	0.97	0.98
f-easure	0.73	0.98	0.98

Table 4: Comparing BLProM with OWASP ZAP in scanning WackoPicko

Approaches / Criteria	BLProM	OWASP ZAP	improvement compared to OWASP ZAP (%)
# GraphNodes	22	89	75.2
#GraphEdges	48	270	82.2
# process (P)	12	48	75
Average edge in each process ($\overline{E}$)	4	46	91.3
Average edges in all processes (P*$\overline{E}$)	48	2208	97.8
#business processes	10	NA	--

Table 5: Comparing BLProM with OWASP ZAP in scanning osCommerce

Approaches / Criteria	BLProM	OWASP ZAP	improvement compared to OWASP ZAP (%)
# GraphNodes	40	170	76.4
#GraphEdges	66	379	82.5
# process (P)	23	17	26
Average edge in each process ($\overline{E}$)	3	113	97.3
Average edges in all processes (P*$\overline{E}$)	69	1921	96.4
#business processes	18	NA	--

Table 6:Comparing BLProM with OWASP ZAP in scanning TomatoCart

Approaches / Criteria	BLProM	OWASP ZAP	improvement compared to OWASP ZAP (%)
# GraphNodes	66	150	56
#GraphEdges	87	410	78.7
# process (P)	31	39	20.5
Average edge in each process ($\overline{E}$)	4	101	96
Average edges in all processes (P*$\overline{E}$)	156	3131	95
#business processes	30	NA	--

Table 7: Comparison of the average of BLProM and the average of ZAP approach in the scanning of selected web applications

Approaches / Criteria	BLProM	OWASP ZAP	improvement compared to OWASP ZAP (%)
# GraphNodes	42.6	136.3	69.2
#GraphEdges	67	353	81.1
# process (P)	20	36.6	40.5
Average edge in each process ($\overline{E}$)	3.6	86.6	94.8
Average edges in all processes (P*$\overline{E}$)	91	2420	96.4

BLProM goal is to identify the business layer of the web application. We can identify business logic vulnerability by identifying business layer of the web application. BLProM detects 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.
In order to evaluate proposed approach, we first show the accuracy of clustering by the following criteria:

- True Positive: Samples that fit well in to 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:

$$recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \quad (1)$$

- Precision: It is calculated by the following formula:

$$precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \quad (2)$$

- F-Measure: It is calculated by the following formula:

$$F - Measure = \frac{2 * recall * precision}{recall + precision} \quad (3)$$

The value of these criteria for ClusteringWebPages is shown in Table 3. In the first row, *#samples* show total number of web pages in HTTP traffic. In the second row, *#clusters* show total number of clusters. In the next rows, the criteria listed above are calculated for each web application.
In order to evaluate the proposed approach, we compare the output of BLProM with scanning output of OWASP ZAP for selected web applications. BLPRoM output for WackoPicko is shown in Table 4. *#graphNodes* show total number of clusters. *#graphEdges* is total number of edges. *#process* is total number of paths from first node. average edges in each process ($\overline{E}$) is calculated by sum of edges of each process divided by total number of processes. An average edge in all process is calculated by product of total

number of processes (P) in average edges of each process ($\overline{E}$). *#business processes* is total number of business processes in the web application.

Existing values in the first column of Table 4 are the output obtained from the proposed approach. The values in the Second column are the scanning output of ZAP. The third column shows the percentage of scanning improvement of our proposed approach compared to the ZAP scan. The results of the table indicate that ZAP scan is a non-smart scan. As a result of this non-intelligent scan, ZAP is not able to identify business layer vulnerabilities.

BLPRoM output for osCommerce is shown in Table 5. The third column shows the percentage of scanning improvement compared to ZAP scanning. It is observed that the web application scanning is improved by identifying web application business layer.

BLPRoM output for TomatoCart is shown in Table 6 . The third column shows the percentage of scanning improvement of our proposed approach compare to ZAP scanning.

Table 7 shows the average of proposed approach, average of OWASP ZAP and the average percentage of improvement in the scanning of selected web applications. For example, in the "Average edges in all processes" benchmark, our approach has been improved by about 96 percent compared to OWASP ZAP.

According to the results presented in tables can be clearly observed that BLProM has improved web application scanning. BLProM is aware of web application business processes. By identifying web application business layer, web scanners can detect business layer vulnerabilities.

V. CONCLUSION

Business logic vulnerabilities are strong vulnerabilities that compromise web application security. Web scanners cannot detect business logic vulnerabilities because they are unable to understand business logic of the web application. For detecting business logic vulnerabilities, business logic of the web application needs to be understood. Therefore, these vulnerabilities are specific to the application and difficult to identify.

In this paper, we proposed BLProM, a black-box approach for detecting business processes of the web application. The aim of BLProM is to ease detecting business logic vulnerabilities. BLPRoM output is used as input in dynamic security testing of the web applications in the business layer in order to detect business logic vulnerabilities. BLProM consists of two main steps: 1-Extracting user navigation graph 2-Detecting web application business processes.

At lab we scan three web applications by BLProM and OWASP ZAP, an open source web application. We showed that BLProM improved scanning about %96 compared to OWASP ZAP. BLProM improved scanning of web application because it clusters web pages prevent scanning similar web application pages.

REFERENCES

[1] Common vulnerabilities and exposures. URL https://cve.mitre.org/cve/cve.html

[2] Identity Theft Resource Center Breach Report Hits Near Record High in 2015, ITRC, 2015. Accessed Feb, 2017. http://www.idtheftcenter.org/ITRC-Surveys-Studies/2015databreaches.html

[3] G. Pellegrino and D. Balzarotti, "Toward Black-Box Detection of Logic Flaws in Web Applications," presented at the *NDSS*, Canada, USA, 2014.

[4] Testing for business logic, OWASP. Accessed Feb, 2017. https://www.owasp.org/index.php/Testing_for_business_logic

[5] D. Balzarotti, M. Cova, VV. Felmetsger and G. Vigna, "Multi-module vulnerability analysis of web-based applications," *CSS*, Virginia, USA, Oct. 28, 2007, pp. 25-35.

[6] Doupé et al., "Fear the EAR: discovering and mitigating execution after redirect vulnerabilities," *CSS*, Chicago, USA, Oct. 17, 2007, pp.251-262.

[7] V. Felmetsger et al., "Toward automated detection of logic vulnerabilities in web applications". *USENIX Security*, Washington, USA, August 11 - 13, 2010, pp.10-10.

[8] Wang T. "The design and implement of the decision support systems of logistics distributing center based on XML". IEEE International Conference on InAutomation and Logistics, pp. 582-586, 2009.

[9] X. Li and Y. Xue, "BLOCK: a black-box approach for detection of state violation attacks towards web applications," *ACSAC*, Florida, USA, Dec. 05 - 09, 2011, pp.247–256.

[10] M. Cova et al., "Swaddler: An Approach for the Anomaly-Based Detection of State Violations in Web Applications", *RAID*, Atlanta, USA, Sep. 18-20, 2007, pp.63–86.

[11] M. Alidoosti and A. Nowroozi, "BLDAST: business-layer Dynamic Application Security Tester of the web application in order to detect web application vulnerabilities against flooding DoS attacks". Iran Society of Cryptology Conference, shiraz, Iran, 2017, in Persian

[12] Alidoosti, A. Nowroozi, "BLTOCTTOU: business-layer dynamic application security tester of the web application in order to detect web application vulnerabilities against Race Condition attacks" Computer Society of Iran Conference, Tehran, Iran, 2018, in Persian. http://csi.org.ir/fa/paper/view/id/2531