Automatic Detection of Access Control Vulnerabilities in Web Applications by URL Crawling and Forced Browsing

Ho-Gil Song[1,2], Yukyong Kim[2] and Kyung-Goo Doh[2⋆]

[1] SureSoft Technologies, Inc., Seoul, Korea
hgsong@suresofttech.com
[2] Hanyang University ERICA, Ansan, Korea
doh@hanyang.ac.kr

Abstract. Access control vulnerabilities can be disastrous in Web applications. The vulnerabilities might be introduced when developers set up unsafe policies in design phase or inconsistently implement safe policies. Attackers take advantage of the vulnerabilities to obtain the authority of administrator and the sensitive information of another user. Hence, the early detection of access control vulnerabilities is very important. This paper proposes a dynamic analysis that automatically detects access control vulnerabilities in web applications. Given a web site and authorities, accessible URLs for each authority are collected by crawling the web site, and then a chosen subset of the URLs are tested to check whether or not access control vulnerabilities exist for the given authority. We implemented the idea, experimented it with some selected web applications, and found some real access-control vulnerabilities.

Key words: software security, access-control, security vulnerabilities, dynamic analysis, Web applications, Web crawling

1 Introduction

Web applications often provide some privileged access only to authorized users. Access control policies, which tell who has privileged access to which resources, are determined and specified at design phase. Web application developers then should carefully write the code making sure that they consistently follow the policies. However, it is often the case that access control policies are not properly set up or not consistently implemented, causing access control vulnerabilities in web applications. Unauthorized users are able to make use of the vulnerabilities

⋆ Corresponding author. This work was supported by the Engineering Research Center of Excellence Program of Korea Ministry of Education, Science and Technology (MEST) / National Research Foundation of Korea (NRF) (Grant 2012-000046), and (Grant 0004-20211) Business for Cooperative R&D between Industry, Academy, and Research Institute funded by Korea Small and Medium Business Administration in 2011.

to illegally escalate their privilege. Hence it is important that developers validate that the implementation does not violate access control policies, so that privilege escalation attacks are prevented. Unfortunately, however, due to the difficulty in designing access control policies and implementing perfect access control checks, web applications are often vulnerable to access control attacks.

In this paper, we present a dynamic analysis technique that automatically detects access control vulnerabilities in Web applications. Given a web site and authorities, accessible URLs for each authority are collected by crawling the web site, and then a chosen subset of the URLs are forced-browsed to check whether or not access control vulnerabilities exist for the given authority. The experiment shows that the implementation actually found some access control vulnerabilities, namely privilege escalation, in real-life Web applications. The method does not have false positive by nature.

This paper is organized as follows: after explaining the automatic detection method and algorithm in Section 2, and concluding in Section 3. The full version of the paper will include a section of related works and experimental results.

2 Methodology and algorithm

2.1 URL crawling

URL crawling is a process of finding in an orderly fashion all the hypertext links (URLs) and collecting accessible ones residing in a given Web site. The URLs are exposed in Web pages as attribute values in elements listed in Table 1. The

Elements	Attributes	Elements	Attributes	Elements	Attributes
a	href	form	action	applet	codebase
area	href	object	archive	object	data
link	href	body	background	img	longdesc
base	href	blockqute	cite	frame	longdesc
frame	src	q	cite	iframe	longdesc
iframe	src	del	cite	head	profile
script	src	ins	cite	img	usemap
input	src	object	classid	input	usemap
image	src	object	codebase	object	usemap

Table 1. Attributes containing URL values

process first finds all the URL strings from the root page of a given site, sends the HTTP request using GET method for each URL, and then saves the URLs of accessible response pages. The process goes on recursively for all the saved pages until no more URLs are found. The detailed URL crawling algorithm is as follows:

```
algorithm url-crawler(url):
```

```
response = http-get-request(url)
if response is OK:
    url-set = {url}
    for u in collect-url(response)
        u = build-valid-url(url,u)
        url-set = set-union(url-set, url-crawler(u))
else:
    url-set = {}
return url-set
```

2.2 Forced browsing

When a Web application employs an access control policy and gives different authorities to users depending on the policy, the URLs collected can be different depending on the given authority. URL crawling with different authorities, say A_1 and A_2, may result in different URL sets, say P_1 and P_2, respectively. The intersection of two URL sets, $P_1 \cap P_2$, represents the pages that are accessible with either authority. The set difference, $P_1 - P_2$, represents the pages that are only accessible with authority A_1. Similarly, $P_2 - P_1$ represents the pages that are only accessible with authority A_2. If we can successfully access either $P_1 - P_2$ with authority A_2 or $P_2 - P_1$ with authority A_1, we can say that there exist privilege escalation which is considered as access control vulnerabilities.

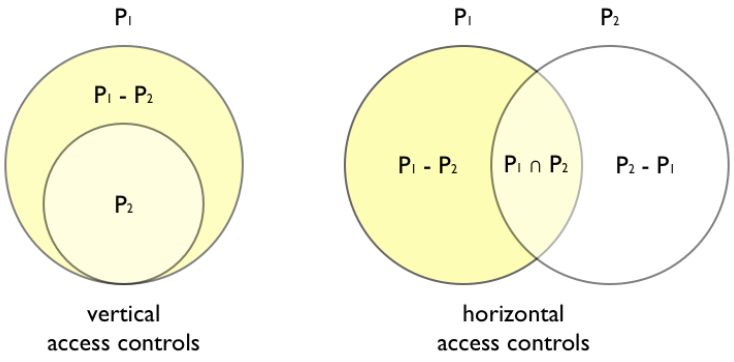

Fig. 1. Categories of access control: vertical and horizontal

Access control policies can be broadly categorized into vertical and horizontal access controls [8]. Vertical access controls typically involve a division between ordinary users and administrators as shown in the lefthand side of Figure 1. Say that an ordinary user with authority A_2 can only access pages in P_2 and an administrator with authority A_1 can access pages in P_2. Thus the pages in $P_1 - P_2$ are only accessible to the administrator. Horizontal access controls allow users to

access a certain subset of a wider range of resources of the same type as shown in the righthand side of Figure 1. For example, a mail application may allow you to read your email but no one else's. Pages in $P_1 - P_2$ are only accessible to a user with authority A_1, but not to the one with A_2. Thus we can detect the existence of privilege escalation by *forced browsing* pages in $P_1 - P_2$ with authority A_2 and analyzing the response. If the response is the same as the one obtained with authority A_1, the page is access control vulnerable, resulting privilege escalation. Note that there is no $P_2 - P_1$ in vertical access controls. In real-life applications, vertical and horizontal access controls are intertwined. In that case, we can still apply the above methodology and obtain the difference set. If there are more than three different authorities, every combination of different pairs has to be tried. There are ${}_nC_2$ combinations to try when there are n different authorities.

The algorithm for detecting privilege escalation is as follows:

```
algorithm detect-privilege-escalation(url,authority1,authority2):
    vulnerable-pages = empty-set
    get-access-privilege(authority1)
    url-set1 = url-crawler(url)
    get-access-privilege(authority2)
    url-set2 = url-crawler(url)
    url-set = set-difference(set-union(url-set1,url-set2),
                             set-intersection(url-set1,url-set2))
    for u in url-set:
        get-access-privilege(authority1)
        response1 = http-get-request(u)
        get-access-privilege(authority2)
        response2 = http-get-request(u)
        if response1 == response2:
            vulnerable-pages = set-union(vulnerable-pages,{u})
    return vulnerable-pages
```

For Web applications that require user authorization, forced browsing should be accompanied by authorization request. The information for authorization can be retrieved from `form` element.

3 Conclusion

In this paper, we propose a method of detecting access-control vulnerabilities in Web applications by forced browsing URLs automatically collected by dynamic URL crawling. The method proposed in this paper detects access-control vulnerabilities (privilege escalation to be exact) by forced browsing using URLs automatically obtained by Web crawling technique. Neither source code nor access-control policies is required for the detection. Furthermore, even without administrator's authority, the method can detect horizontal privilege escalation between users as well as vertical privilege escalation between public and authorized users. The method produces no false positive because it applies forced

browsing only to dynamically crawled pages. However, there might be some false negatives since URLs collected by dynamic browsing comprise the entire resources the Web application has. There is a room for improvement in URL crawling. We might be able to find more URLs in the client-side scripts and HTML documents.

References

1. Fisler, K., Krishnamurthy, S., Meyerovich, A., Tschantz M.C.: Verification and change-impact analysis of access-control policies. In Proceedings of the 27th International Conference on Software Engineering, ICSE'05, pp. 196–205. (2005)
2. Jayaraman, K., Ganesh, V., Tripunitara, M., Rinard M., Chapin, S.: Automatic error finding in access-control policies. In Proceedings of the 18th ACM Conference on Computer and Communication Security, CCS'11, pp. 163–174. (2011)
3. Kang, W.J.: A Method for Access Control on Uncertain Context. The Journal of IWIT (The Institute of Webcasting, Internet and Telecommunication). vol.10, No.2, pp.215–223. (2010)
4. Kang, W.J.: A Method for Semantic Access Control using Hierarchy Tree. The Journal of IWIT (The Institute of Webcasting, Internet and Telecommunication). vol.11, No.6, pp.223–234. (2011)
5. Kolovski, V., Hendler, J., Parsia, B. : Analyzing web access control policies. In Proceedings of the 16th International Conference on World Wide Web, WWW'07, pp. 677–686. (2007)
6. Martin, E., Xie, T.: A fault model and mutation testing of access control policies. In Proceedings of the 16th International Conference on World Wide Web, WWW'07, pp. 667–676. (2007)
7. Raghavan, S. and Garcia-Molina, H.: Crawling the hidden web. In Proceedings of the 27th International Conference on Very Large Data Bases, VLDB'01, pages 129-138 (2001)
8. Stuttard, D., Pinto, M.: The Web Application Hacker's Handbook: Discovering and Exploiting Security Flows. Wiley (2007)
9. Sun, F., Xu, L., Su, Z.: Static detection of access control vulnerabilities in web applications. In Proceedings of the USENIX Security Symposium. (2011)
10. Zhang, N., Ryan, M., Guelev, D.: Evaluating access control policies through model checking. In Information Security. LNCS, vol. , pp. 446–460. Springer, Heidelberg (2005)