**AURLS：An Active Defense Solution for Web Applications Using URL Shifting**

## Abstract

Web security is important for web applications as they are widely used in various fields. In this paper, we proposed a web active defense solution combining URL shifting and user identity binding, called AURLS, to achieve "one URL, one user". As an improvement, we further add the URL whitelist technology and fine user access management. Compared with the traditional defense schemes, our solution increases the difficulty for constructing attack, achieve a more accurate user access management and avoid false detection. To verify the proposed solution, we execute comparative experiments on the DVWA and other systems to evaluate the defense effect of AURLS. The results show that AURLS can effectively prevent many high-risk attacks, such as brute force attack, XSS, and phishing, with little interference in user experience and insignificant loss of system performance.

## 1 Introduction

Web applications are widely used in various fields. Besides email, news sites and other traditional applications, many mission-critical tasks are also delivered by web applications, such as e-finance, e-commerce, e-health, e-government. Openness of web applications not only promotes their wider use but also leads to increasingly prominent security issues. Currently, websites are the main targets of hackers, and often suffer from many attacks, such as data tampering, SQL injection, cross-site scripting, command insertion, and buffer overflow. Report [[1](#_ENREF_1" \o "Consortium, 2008 #88)] indicates that 49% of the web applications reviewed contain vulnerabilities considered high risk. A 2009 SANS study found that attacks against web applications constitute more than 60% of the total attack attempts observed on the Internet.

Many techniques have been proposed to secure web applications. They are organized under three categories.

(1) Secure construction of new web applications: This class of technique aims to construct a secure web application by defensive coding approaches to ensure that no potential vulnerabilities exist in the application [2-6].

(2) Security analysis/testing of legacy web applications: This class of technique verifies the security properties held by a web application and the vulnerabilities within the application, and is also known as vulnerability analysis.

(3) Runtime protection of legacy web applications: This class of technique aims to protect a potentially vulnerable web application against attacks by building a runtime environment that supports its secure execution.

This paper focuses on the research of the runtime protection method, and aims to improve the following shortcomings of the existing methods.

(1) The existing methods are mainly based on rules, and have poor defense effectiveness for new vulnerabilities (or zero-day vulnerabilities).

(2) The management of the user's access is mainly based on the IP address, which leads to the management granularity too course. In addition to reducing the detection effect, it sometimes leads to false positives.

(3) The access URL for a system resource is consistent with all users, and it is fixed, so that: (a) the attacker can scan the network topology of the system to find out the vulnerabilities; (b) the attacker can determine the access address of other users, making it convenient for the attacker to construct the attack code.

In this paper, inspired by the idea of one-time-pad, we proposed a web active defense solution based on URL shifting, namely AURLS. This solution dynamically changes the access URL of the system resource, and binds the virtual URL to the user identity to realize "one URL, one user". We further joined the URL whitelist technology and finer user access management to improve system defense capability. Compared with current defense technologies, this method has the following advantages:

(1) It is difficult for an attacker to obtain the network topology of the system by scanning, which can effectively reduce the attack surface for server, and make it hard for attackers to construct attack codes.

(2) The attacker can't obtain URLs for other users. It can prevent attacks against other users, such as reflective XSS, CRSF, phishing links, and so on.

(3) Because each visit can be accurate to a specific user, we can establish more precise model of web access and perform user access management more finely, to protect against XSS, brute force crack, path traversal, etc.

(4) The protection is not based on the rules, so this method can defend against attacks through unknown vulnerabilities to a certain extent.

The remainder of this paper is organized as follows. In section 2, we introduce the related work in web security on the server side. In section 3, we describe the AURLS solution in detail. In section 4, we evaluate the proposed defense mechanism in several systems and discuss these results of the experiments. Finally, we draw conclusions in section 4.

.

Fig. 1 Architecture of AURLS

## 2 Related works

Web application security involves both the client and server sides. In this paper, we narrow our research scope and focus on approaches that are deployed on the server side. Web security technologies for the server side include the traditional protection technology, the MTD technique, the URL rewriting method, etc.

(1) Secure Construction Techniques

Study [2] defines a set of general software security coding specifications that can be integrated into the software-development life cycle to reduce the most common software vulnerabilities. In addition, OWASP also recommended a number of web application development processes involved in the common norms. Study [3] lists a number of C and C++ applications in development and the relevant security coding specifications, and shows that the incorrect coding may bring security issues.

(2) Security Analysis Techniques

In this kind of method, attention should be paid when choosing a proper assessment tool. Curphey and others recommend eight tools [4], including HTTP proxies, web application (black-box) scanners, and database scanners. They also suggest how to select the right tool, with the advantages and disadvantages concerned. Felmetsger, V. et al. proposed an automated detection of application logic vulnerabilities [5]. They developed a tool, Waler, and applied it to a number of web applications to find previously-unknown logic vulnerabilities. Waler infers specifications that partially capture a web application’s logic and analyzes the inferred specifications with respect to the web application’s code and identify violations. Jovanovic, Kruegel et al. address the problem of vulnerable web applications by static source code analysis. They present Pixy, a tool for statically detecting XSS and SQL injection vulnerabilities in PHP 4 codes by data flow analysis [6]. Shar et al. proposed an inspiring approach to removing XSS vulnerability by developing a tool, saferXSS [7].

(3) Runtime Protection Techniques

Many of the techniques used for runtime protection of legacy applications only address one particular type of attack [8, 9]. Razzaq et al. evaluate different web application firewalls through some necessary features [10]. Instead of detecting the most well-known web vulnerability such as in [11-13], WebDefender [14] is devoted to detecting those attacks based on input validation through pattern recognition, demonstrating good performance with flexibility and expandability. In study [13], a negative security model is presented to detect and prevent common attacks. Based on this, a rule-based Web Application Firewall (WAF) appears, which can mitigate or block almost all existing attacks. Moreover, Tekerek raises a hybrid web application firewall by utilizing signature-based detection and other detection methods [15].

(4) Moving Target Defense Techniques

MTD techniques create moving attack surfaces for web services to introduce diversity and uncertainty. Y. Huang et al applied the MTD concept to constitute a dynamic and uncertain attack surface, including various offline servers rotating to replace online servers [16]. Similarly, M. Taguinod et al developed two MTD approaches to mitigate several classes of web application vulnerabilities and allow for changing language implementation of the web application and database implementation [17]. However, semantic issues still remain [18, 19]. Besides these traditional techniques, novel moving target defense (MTD) techniques are applied to protect web applications [16, 17]. MTD techniques introduce diversity and uncertainty to create moving attack surfaces for web services, including changing the server-side language, and shifting the database into different implementations. MTD techniques increase the complexity and cost for attackers, and reduce the information asymmetry between the attacker and defender. MTD-based techniques can increase the difficulty of attack, but are still in a theoretical research stage because of the difficulty of implementation.

(5) URL Rewriting Techniques

One of the methods that are similar to the proposed algorithm we present in this paper is URL rewriting. URL rewriting can turn unsightly URLs into nice ones — with a lot less agony and expense than picking a good domain name. Salz, Kuznetsov et al. proposed a method of encoding a remote record identifier that maintains compatibility with active content by creating a new identifier from a base portion and a path and/or query portion. They also disclosed a method of decrypting an encrypted rewritten record identifier and a gateway apparatus for mediating communication between a client system and a server system using the remote record identifier encryption and decryption methods. [20, 21].

## 3 The AULS Solution

Our solution can be generally divided into two parts, namely the URL shifting and the URL validity check. The URL shifting is responsible for changing dynamically the URLs in a response and binding these shifting URL with the users' identity, to realize "one URL one user." URL validity checking is responsible to check whether the request is valid according to the information stored in the system.

We describe the proposed solutions in the following, including the system architecture, key technologies, and related processes. The main key technologies involved in this solution include the URL shifting, the binding mechanism of the virtual URL and the user identity, and the user access validation.

## 3.1 The architecture of AULS

As illustrated in Fig.1, AULS system consists of three components: the URL shifting module, the URL checking module, and the URL conversion module.

The URL shifting module replaces all URLs in the response from the web server with virtual URLs generated at random. It also binds the shifting URLs with the user identity, and forwards the response transformed to the client.

The URL validity check module receives the request from the client and examines the validity of this request.

If the request is valid, the URL conversion module translates the virtual URL with the original URL, and forwards this request to the web server.

## 3.2 URL shifting

The URL shifting involves the dynamic URLs generated in the client-side, the static URLs and the dynamic URLs (include the form URLs) generated on the server-side.

The client-side dynamic URLs refer to these URL addresses that generated during page rendering. After web 2.0, there are a number of links generated by the client of the web applications. When the web service is accessed through these URLs, AURLS will block these requests to cause the user to fail to access the application.

Our solution for client-side dynamic URLs is as follows. First, we use a taint analysis method [2-4] to find the web pages that contain the client dynamic URLs, and obtain the information about client's dynamic URL. When a response is received, AULRS first checks whether this response contains some client dynamic URLs. If it does, AURLS looks up the strings associated with these URLs, and then replace them by strings generated in randomly (a tag and a fixed length string) and make the binding between the shifting strings and user identity. The subsequent URL validity check is based on the information of these shifting strings.

The shifting for static URL and dynamic URL in the server-side are directly replaced by a random string. The shifting for the form dynamic URL in the server-side is similar to the client's dynamic URL string replacement, which also uses a tag and fixed length approach.

## 3.3 The binding mechanism of virtual URL and user identity

The simple URL shifting technique can reduce injection points on a web system, which makes it difficult for a hacker to forge an attack code. When a response is returned, the simple URL shifting method replaces all the URLs it contains with the randomly generated URLs and make mappings between the virtual URLs[[1]](#footnote-1) with the real URLs.

However, because these virtual URLs are not associated with the user, the URLs in a page will be shifted again when the page is revisited. It will cause a lot of redundant mappings of virtual URL and real URL in the system. This will greatly reduce the system processing efficiency and increase the page access time, resulting in a serious deterioration of user experience of web access. In addition, after the attacker steals cookies from other users, the attacker can get a valid shifting URL for subsequent attacks.

To solve the above problem, we propose a binding mechanism to associate the user virtual identity[[2]](#footnote-2) with the virtual URLs. By this technique, for the same user, the shifting URL is only generated once for the same URL. If the attacker tries to construct a malicious script and defraud the user's access, like phishing and CSRF, because he or she cannot know the URL belongs to the victim. Thus the URL in the malicious code is not match the victim's identity, this attack will be rejected. The binding mechanism is design as follows:

(1) Each user is identified with a unique token GUID.

(2) Establish the following mappings: a) the mapping of the virtual URL and the GUID; b) the mapping of the virtual cookies and the GUID; c) the mapping of GUID and the pair of the user IP and the virtual cookie; d) the mapping of the virtual cookies and the real cookies.

(3) When an HTTP request is received, AURLS detects whether the URL of this request is virtual.

If the URL of this request is virtual, AURLS compares the cookie in this request to the virtual cookie in the user identity. If they are different, this request is invalid and be blocked. Otherwise, AURLS checks whether the IP of this request is same to the IP in the user identity. If they are different, AURLS redirects this request to the system authentication page.

If the URL of this request isn’t virtual, AURLS checks if this URL is in the whitelist. If it is not in, AURLS blocks this access. Otherwise, AURLS checks whether there is a cookie in this request. If there is no cookie, this access can be determined as the first visit of this user. If there is a cookie, AURLS obtains the user identity according to the cookie in the request and then checks whether the client IP address of this request is same to the one in the user identity. If they are identical, it's a case of the user accessing the home page again. If they are different, it is possible that the attacker has stolen other users' cookies. AURLS redirects the request to the authentication page.

(4) Maintenance of virtual cookie and real cookie mapping relationships. When the web server returns the latest new cookies, AULRS updates the key-value pairs for the virtual cookie which the user bound.

## 3.4 The check approach for the validity of user access

There are several criteria for determining the request validity. (1) The URL of the request is in whitelist and without a cookie; (2) the URL of the request is in the whitelist and with a cookie, and the cookie and IP address of the request are consistent with the corresponding user; (3) the URL of the request is not in the whitelist, and the cookie of the request is consistent with the corresponding user; (4) No violation of user access policy.

Compared with the access control based on the IP address, each request is corresponding to a specific user while using AURLS, so it is possible to perform a finer user access control. The user access control of AURLS includes the total number of visits, the access frequency, the maximum access interval, the 2-gram access pattern for a specific virtual URL, etc. In the following sections, we introduce the URL shifting procedure, and the URL validity checking and conversion procedure in detail.

## 3.5 Procedure of URL shifting

As shown in Fig. 2, the URL shifting procedure mainly includes consists of the following phases: (1) Receive the response from the web server; (2) Generate or update virtual cookie and real cookies pair mapping; (3) Find actual URLs in the response; (4) Generated virtual URLs randomly and update or set access policy of virtual URL; (5) Replace actual URLs in this response with virtual URL. More detailed steps of URL shifting is are presented below.

**Begin**

Step 1: Receive the response returned by the web server and parse its header to obtain the cookies.

Step 2: Update the cookie field in the virtual cookie (GUID, cookie) table with this cookies.

Step 3: Check whether the page contains the client dynamic URL. If it contains, set the value of flag *client\_dynamic\_url* to be true.

Step 4: Read a character from this web response stream.

Step 5: If a URL is fully read, go to step 6. Otherwise, go to step 7.

Step 6: If this URL is relative, go to step 14. Otherwise go to step 15.

Step 7: Check that the flag *client\_dynamic\_url* value is true and fully read to a string related to a client dynamic URL. If it is true, go to step 8. Otherwise, go to step 22.

Step 8: Find out if there is a corresponding virtual string in the system based on the user GUID and the related string. If no, a random virtual string is generated, and the related string is replaced, go to step 9. Otherwise, go to step 12.

Step 9: Add a triple-tuple (uid, actual\_string, virtual\_string) record into system, then go to step 10.

Step 10: Add a two-tuple (virtual\_string, actual\_string) record into system, then go to step 11.

Step 11: Set the timeout threshold, access frequency limit, and maximum access times of this virtual string, then go to step 13.

Step 12: Update the expiration time corresponding to the virtual URL.

Step 13: Replace the actual string in the response web page with the virtual string, then go to step 22.

Step 14: Transform the relative URL to an absolute URL.

Step 15: Query whether there is a corresponding virtual URL according to the user GUID and the absolute URL. If not found, go to step 16. Otherwise go to step 20.

Step 16: Generate a virtual URL at random (If the URL is a form URL, the rule for generating the virtual URL randomly is tag and fixed length).

Step 17: Add a triple-tuple (uid, actual\_url, virtual\_url) record into the map of user and URL.

Step 18: Add a two-tuple (virtual\_url, actual\_url) to the virtual URL mapping table in the system.

Step 19: Set the timeout threshold, access frequency limit, and maximum access times of this virtual URL, then go to step 21.

Step 20: Update the expiration time corresponding to this virtual URL.

Step 21: Replace the actual URL in this response with this virtual URL.

Step 22: Check whether this response has been processed completely. If this processing is finished, go to step 23. Otherwise, return to step 4 again.

Step 23: Send the web page processed to the client.

**End.**



Fig. 2 Flowchart of URL shifting



Fig. 3 Flowchart of URL validity checking and conversion

## 3.6 Procedure of URL validity checking and conversion

In the URL validity checking and conversion procedure (shown in Fig. 3), AURLS checks the validity of a URL in the user’s request and converts the virtual URL to an actual URL. This procedure consists of five phases: (1) Receive request from client, parse it to extract the URL and the cookie; (2) Check the validity of the URL in this request; (3) Start intrusion response; (4) Replace the virtual URL in this request with the corresponding actual URL; (5) Forward the request processed to the web server. This procedure is described in detail as follows:

**Begin.**

Step 1: Receive a request, and extract the URL and the cookie in this request.

Step 2: Check if the URL is a valid virtual URL dynamically generated in the client side. If not, go to step 3. Otherwise, go to step 14.

Step 3: Check if the URL is a valid form virtual URL. If not, go to step 4. Otherwise, go to step 14.

Step 4: Verify that the URL is a valid server dynamic virtual URL. If not, go to step 5; otherwise, go to step 14.

Step 5: Check if the URL is a valid static virtual URL. If not, go to step 6. Otherwise go to step 7.

Step 6: Block this request and start intrusion response, and go to **End**.

Step 7: Check if there is a cookie in the request. If a cookie is in this request, go to step 8. Otherwise, step 22.

Step 8: Look up the GUID of user according to this cookie carried by the request. If not found, go to step 6. Otherwise, go to step 9.

Step 9: Obtain the IP address according to this GUID of the user, and then check if the IP address is corresponded to the IP address of the request. If it is not corresponded, it may be that the attacker misappropriates the user's virtual cookie, goes to step 10. If it is corresponding, that is the user visit the home page again, go to step 13.

Step 10: Re-authenticate user. If user authentication is successful, go to step 13; otherwise, go to step 6.

Step 11: Generate a virtual cookie for the user, and add a mapping of a virtual cookie with a real cookie (null value) to the system.

Step 12: Create a new GUID for the user and add a map of this GUID and the user's virtual identity (GUID, <virtual\_cookie, IP>) to the system.

Step 13: Forward this response to the web server, and go to **End**.

Step 14: Check whether the virtual cookie associated with the user in the system is the same as the one in this request. If they are identical, go to step 15. Otherwise, go to step 6.

Step 15: Check whether the virtual URL is expired (This can be determined by how much time the URL is not accessed. If it is expired, go to step 10. Otherwise, go to step 16.

Step 16: Check the cumulative times of accesses to this virtual URL. If it exceeds the limit value, go to step 10. Otherwise, go to step 17.

Step 17: Check the access frequency for this virtual URL. If it exceeds the limitation, go to step 10. Otherwise, go to step 18.

Step 18: Check whether the *Referer* field for the request is in the system's virtual URL list or in the URL whitelist. If not found, go to step 6. Otherwise, go to step 19.

Step 19: Replace the virtual URL and the *Referer* field in the request with the actual URLs.

Step 20: Check whether the virtual URL of the request and the *Referer* field is in the system's 2-gram access behavior pattern table. If not, go to Step 6. Otherwise, go to step 21.

Step 21: Update the latest access time of this URL, and go to step 14.

**End.**

## 4 Experiments and Discussion

In this section, we evaluate the solution presented in this paper. The test environment is shown in Fig.4, which consists of an AURLS system, a web server (192.168.1.149), an attacker’s server (192.168.1.150), and multiple terminals of client. We evaluate the AURLS solution in terms of three aspects: the protective effect for the web system, compatibility, and the impact on the performance of the web application. The protective effect is measured by what vulnerabilities the AURLS technology can prevent. The indexes of compatibility test include page display consistency, page function consistency, script compatibility, layout consistency and so on. The indexes for the web system’s performance include average response time for accessing web page, and throughput of the web server. In the following section, we describe the evaluation experiment in detail.

## 4.1 Experiments on defense effectiveness test with AURLS

In the system protection experiment, we chose the Damn vulnerable web application (DVWA) [[2](#_ENREF_28" \o ",  #79)7] as the benchmark system. DVWA is a PHP/MySQL web application that is very vulnerable. It has many common web vulnerabilities built-in, including the Top 10 vulnerabilities of the OWASP.

For testing the defense effect of AURLS, we set the security level of the DVWA to *LOW*, and did not install any other protection tools besides AURLS in this experiment. We conducted the penetration test without AURLS and with AURLS. The test methods included manual testing and automated tools.

The tests for DVWA included brute force attack, CSRF, file inclusion, XSS (reflected) and XSS (stored). The detail of the test procedures and results are described as follows.



Fig. 4 Test environment

## 4.1.1 Brute Force Attack

Since DVWA does not apply verification code technology, we carried out password cracking by a brute force attack.

In the case without AURLS, we used the burpsuite tool and soon figured out one user account and password (username: “admin”, password: “password”).

In the case with AURLS, we perform this attack again. Apart from several requests getting normal responses at the beginning, the rest all returned a 403 error. Obviously, the attack failed. The successful defense is attributed to the virtual URL and user identity binding technique.

## 4.1.2 CSRF

In this test, we designed a case to change a password without the victim’s knowledge. The prerequisite of the attack is that the victim is visiting or has visited the password change page and the cookie for this session is stored in the user’s browser. Assuming the link to change the password is “http://192.168.1.149/vulnerabilities/csrf/?password\_new=hacker&password\_conf=hacker&Change=Change”. The attacker induces the victim to visit a phishing link. For example, the attacker sends this victim a phishing link http://192.168.1.150/shopping.html (as shown in Fig.5). If the victim clicks on the link, the page will load a hidden image whose source points to “*http://192.168.1.1*49*/vulnerabilities/csrf/?password*\_*new=hacker&password*\_*conf=hacker&Change=Change”*.

In the case without AURLS, the browser of this victim will automatically and silently access this link to change the password. By checking the server, we found that password had been modified.

In the case with AURLS, it is difficult for an attacker to know the link to change the password. For example, the *italic* font part of the link http://192.168.1.149/ ***048bba5-d147-4bdd-aed2-17cc8eda9f91***?password\_ new=hacker&passwprd\_conf=hacker&Change=Change is unpredictable, so the attack doesn’t work. If a virtual URL corresponding to an attacker is used, it will be detected by the identity binding mechanism and blocked. We carried out the attack again, and found that the password hadn’t been modified.

**<!doctype html>**

**<html lang="en">**

**<head>**

**<style>**

**img{**

**display:none;**

**}**

**</style>**

**</head>**

**<body>**

**<img src="http://192.168.1.149/vulnerabilities/csrf/?password\_new=hacker&password\_conf=hacker & Change=Change">**

**<h1>404</h1>**

**<h3>Not found </h3>**

**</body>**

**</html>**

Fig. 5 Source of the phishing page

## 4.1.3 XSS (reflected)

In the case without AURLS, when entering “</ pre> <script> alert ('xss') </ script> <pre>” in the *name* textbox at the page “http://192.168.1.149/vulnerabilities/xss\_r/vulnerabilities/xss\_r/”, we can trigger XSS (reflected) vulnerability. The attack URL is http://192.168.1.149/vulnerabilities/xss\_r/?name= %3C%2Fpre%3E%3Cscript%3E alert %28 %27 xss%27%29%3C%2Fscript%3E%3C%2F pre %3E#. It can also be triggered when the link is sent to the victim.

However, in the case with AURLS, the attack link is as follows: http://192.168.1.149 /816086c-fd4a-4576-bc70-a06be110bb49?name = %3C%2Fpre%3E%3C script %3E alert %28 % %27%29%3C%2Fscript%3E%3Cpre%3E. Because the virtual URL is bound with the attacker, the link cannot be used for attack. Even if it sent to other user, it does not result in an attack effect.

## 4.1.4 XSS (stored)

In the XSS defense, we conducted two experiments: (1) using the script to change the password; (2) attack after the attacker steals user virtual cookie and accesses the URL in whitelist with this virtual cookie.

(1) Experiment using XSS (stored) to change the password

In the experiment, we input text “<img src= http://192.168.1.149/vulnerabilities/csrf/?password\_new=hacker&password\_conf=hacker&Change= Chan ge alt=“hello”>” in the *Message* textbox at page *http://192.168.1.149/ vulnerabilities/xss\_s/*. This code create an *img* tag which source points to the link to change password. After this script is loaded, the browser will access the link to change the password of a victim.

In the case without AURLS, after the page with the XSS (stored) has been visited, the browser loads this script and change the password.

In the case with AURLS, these steps are the same, but the result is different. Although the script is still loaded and the link to change the password is also triggered. But it was intercepted because the visit violated the 2-gram access pattern, AURLS returned a 403 error to declare the failure of password modification and recorded the access behavior (the administrator can quickly locate the malicious code stored on the server by viewing the intrusion log).

(2) Experiment using the stolen virtual cookie to access the URL in whitelist for attack

In this experiment, we first inject the script "< script > document. Write (' <img SRC = "http://192.168.1.150 /getcookie.PHP?Cookies ='+document.cookie + '/>');" to get the attacker's cookie through the XSS (stored) vulnerability, and then use the cookie to access the homepage of website at http://192.168.1.149/.

In the case without AURLS, the access direct jump to the vulnerability demo page shows that the system cannot effectively defend against such attacks.

However, in the case of the AURLS defense, the access is redirected to the authentication page at http://192.168.1.149/login.php. It shows that the attacker can't effectively launch such attacks after stealing user cookies.

## 4.1.5 File inclusion

In the case without AURLS, we modified the URL to trigger this vulnerability in the page of file inclusion. For instance, we changed the original URL “http://192.168.1.149/dvwa/vulnerabilities/fi/?page = file1.php” to “http://192.168.1.149/dvwa/vulnerabili ties/fi/?page=http://192.168.1.150/oi.php.decode.txt”. The oi.php.decode.txt file is actually a webshell file, which is stored in the remote server (192.168.1.149).

The file inclusion vulnerability is triggered by changing the parameters of the URL. In the case with AURLS, the original URL is encoded as “http://192.168.1.150/2b812669-7b71-49f5-97bd-f8650-1714680” and difficult to modify. We access the page at “http://192.168.1.149/dvwa/vulner-abilities/fi/?page=http://192.168.1.150/oi.php.decode.txt” again and receive a 403 error. The reason is that URL is not a valid shifting address bound to that user and not in the URL whitelist.

### 4.1.6 Summary

In this section, we give a summary of attack types, implementation mechanism, and a detailed analysis of the reasoning in the Table 1.



From the above experimental results, we can conclude that:

AURLS method converts dynamic links in a web page to static links. It significantly reduces the system injection point.

AURLS method significantly improves the system security because the virtual link returned for the different users with the same dynamic link is unique, and the attacker cannot guess other user's links. Obviously, the AURLS method can be employed to defend against CRSF, reflection XSS attacks, etc.

AURLS can prevent against brute force attack thanks to the virtual URL and user identity binding technology, which makes it possible to perform user behavior management more precisely.

AURLS can prevent almost all attacks through the URL generated on the server side. However, if the request is with the form parameters entered by clients, such as the form submission. But in general，AURLS technology effectively limits web attack in a smaller range (attacks generated by the client-generated URL). To fend off this problem, we have combined the simple keyword filter technique to prevent attacks in the practice of protecting a website of our University.

### 4.2 Compatibility testing and evaluation of the impact of system performance

**4.2.1 Compatibility testing**

It includes the integrity test of page display, the integrity test of page function, script compatibility test, layout test, etc.

In the experiment, we access the same web page through the client request in the same test environment, in the cases with and without AURLS, and check the differences between them. Although there are some different in the page source, they are consistent in the integrity of page display, the integrity of page function, the script compatibility and the integrity test of page layout.

**4.2.2 The page access time experiment**

In the case of with AURLS and without AURLS defense, the home page of our university website is accessed 100 times (each of which includes 51 requests) and then the average page access time is calculated separately. The experimental results are shown in figure 6. The page access time are within 2 seconds both the case with AURLS and without AURLS, and all are satisfactory according to the well-known 2-5-8 website usability principles.

**4.2.3 Performance overhead**

To assess the performance impact of web applications in practical usage scenarios, we conducted a load test and a stress test on the homepage of a website of our University(with 45 requests) with the pylot tool [[4](#_ENREF_4" \o "Goldberg, 2009 #81)]. The web server setup consisted of a standard desktop computer with Intel® i7 4770HQ 2.2G CPU and 2GB RAM, running Windows XP OS, Apache 2.2.3 web server with PHP 5.3.28. The AULRS system setup consisted of a standard desktop computer with Intel® i7 4770HQ 2.2G CPU and 4GB RAM, running Ubuntu Server 16.04 OS, Nginx 1.81 server.

We tested three accessing cases (on direct accessing, on through proxy accessing, and on the AURLS solution). In the case with AURLS, we remove the limit of the maximum number of access times. In the load test, the metrics are the average response time, and the throughput (req/sec). The number of concurrent users is varied from 50 to 400(the request interval is 5 seconds). Each test was ran for 3 minutes.



Fig. 6 Page access time with AURLS defense and without AURLS defense (visit home page, total 51 requests)

 

1. Through(req/sec) in three access ways (b)Average response time(sec) under three access ways

Fig. 7 Performance changes under load testing

Table 2 Maximum throughput test on 100 concurrent users, no access interval, lasting access for 1 minute

|  |  |  |  |
| --- | --- | --- | --- |
| Access method | direct | through reverse proxy | AURLS |
| Total number of requests | 4269 | 4197 | 4148 |

Fig 7(a) shows the throughput under the three access conditions. In the three cases, the through per second of the system is basically the same, especially in the case with fewer concurrent users. Fig 7(b) shows the average response time under the three access conditions. The average response time is increasing after using the proxy server or AULRS defense. From Fig 7(b), we can find that increasing in the average response time in the AURLS is caused by the use of the proxy server, not the URL shifting process and validity check.

In the stress test, the number of concurrent users in each test was 400. From the test results shown in Table 2, we can see that the loss of system performance caused by AURLS method is insignificant. Compared with the direct access, the performance loss by using AURLS is less than 3%. Compared with the proxy way, the performance loss causing by AURLS is less than 1.7%.

Both results of load test and stress test show that the AURLS solution does not have a significant impact on system performance.

## 4.3 Discussion

## 4.3.1 The defense effectiveness of simple URL shifting

The simple URL shifting method transforms the URLs in the web page by virtual and static URLs generated randomly. This approach reduces injection points on the web system. However, it has several shortcomings:

(1) It cannot prevent the attack using an URL without parameters, such as a webshell in the form of \*.php.

(2) It cannot associate access with user resulting in the inability to perform precise user behavior management.

(3) It cannot prevent those attacks initiating with virtual URL is obtained by the attacker through a visit to the website.

## 4.3.2 The defense effectiveness of the virtual URL and user identity binding mechanism

The binding mechanism of virtual URL and user's identity addresses the second and the third of the above shortcomings. This technique is applied to identify the same user’s requests. By this technique, we can not only effectively allow the precise management of user behavior, but also prevent the hacker using the known virtual URL attack after the theft of others’ cookie.

## 4.3.3 The defense effectiveness of the URL whitelist

The URL whitelist addresses the first of the above shortcomings. Whitelist is always applied in a simple application environment. In general, web applications belong to a complex application environment and can only apply blacklist. Because of the application of URL shifting, and URL binding mechanism, each request can accurately associate with a user so making it practical for whitelist technology. By the whitelist, we cannot only defend against attack using a URL without parameters and path traversal vulnerability, but also can relieve the system compatibility problem by adding special types of file to the whitelist and improve the friendliness for search engines through adding common access entries to the system.

## 4.3.4 The defense effectiveness of finer user access management.

Compared with the IP based user access management, the AURLS solution can match the virtual URL with the user through the user's virtual identity binding. Thus, AURLS can perform a finer user access management, including setting finer access threshold and restrict user access behavior accurately. It improves the detection rate for intrusion while reducing the rate of false positive.

## 4.3.5 Differences with URL rewriting

URL rewriting means the URL is rewritten into another URL that the site can handle, which has several advantages: (1) Shortening the URL, hiding the actual path means better security. (2) Easy for user to memorize and type. (3) Easy to be included in search engines.

Compared with AURLS, URL rewriting has two drawbacks. Firstly, for different users, the links rewritten are identical (or basically the same). In addition, the URL rewritten links can be reversed.

## 4.3.6 Differences with traditional WAFs

Unlike the traditional WAF, AURLS is not based on certain rules to match attacks, but works by limiting the access entry, the behavior pattern of access, and the frequency and the number of access points to each virtual URL. Compared with traditional WAF, AURLS is a better and universally applicable method for unknown attacks.

## 5 CONCLUSION

In this paper, we proposed a URL address shifting technique to change randomly the URLs in a web page, and combining the identity binding technology to realize "one URL, one user". At the same time, we joined the URL whitelist based on this framework and performed a more sophisticated user access management to improve the defense capability of the system. We also implemented a prototype system on Nginx reverse proxy server, and carried out a series of tests with DVWA system to demonstrate the defense effect for web attacks. Compared to traditional defense technology, we consider that AURLS has the following advantages.

(1) AURLS can reduce the injection points of the site, so that the attacker cannot predict the virtual URL on other terminals, thus increasing the difficulty of attack.

(2) AURLS binds sessions and virtual URL to identify accurately users, which results in a more fine-grained user behavior management.

(3) AURLS is not signature-based, so is more flexible and effective in preventing unknown attacks than traditional methods.

However, there is still much work to be done on AURLS. In the future, we plan to focus on the following aspects: (a) more efficient algorithms for URL lookup and matching; (b) multi-site support; (c) more general intrusion detection methods based on the user access model.

**Reference**

[1] W.A.S. Consortium, Web application security statistics project 2007, in, 2008.

[2] R.C. Seacord, Secure Coding in C and C++, Pearson Education, 2005.

[3] R. Seacord, Secure coding in C and C++ of strings and integers, IEEE security & privacy, 4 (2006) 74-76.

[4] R. Seacord, Top 10 Secure Coding Practices, in, 2011.

[5] N. Antunes, M. Vieira, Defending against web application vulnerabilities, Computer, 45 (2012) 0066-0072.

[6] OWASP, OWASP Secure Coding Practices - Quick Reference Guide, in, OWASP, 2015.

[7] M. Curphey, R. Arawo, Web application security assessment tools, IEEE Security & Privacy, 4 (2006) 32-41.

[8] V. Felmetsger, L. Cavedon, C. Kruegel, G. Vigna, Toward automated detection of logic vulnerabilities in web applications, in: USENIX Security Symposium, 2010.

[9] N. Jovanovic, C. Kruegel, E. Kirda, Static analysis for detecting taint-style vulnerabilities in web applications, Journal of Computer Security, 18 (2010) 861-907.

[10] L.K. Shar, H.B.K. Tan, Automated removal of cross site scripting vulnerabilities in web applications, Information and Software Technology, 54 (2012) 467-478.

[11] F.S. Rietta, Application layer intrusion detection for SQL injection, in: Proceedings of the 44th annual Southeast regional conference, ACM, 2006, pp. 531-536.

[12] D. Kar, S. Panigrahi, S. Sundararajan, SQLiDDS: SQL injection detection using query transformation and document similarity, in: International Conference on Distributed Computing and Internet Technology, Springer, 2015, pp. 377-390.

[13] A. Razzaq, A. Hur, S. Shahbaz, M. Masood, H.F. Ahmad, Critical analysis on web application firewall solutions, in: Autonomous Decentralized Systems (ISADS), 2013 IEEE Eleventh International Symposium on, IEEE, 2013, pp. 1-6.

[14] Y. Xie, S.-Z. Yu, Anomaly detection based on web users' browsing behaviors, Ruan Jian Xue Bao(Journal of Software), 18 (2007) 967-977.

[15] C. Torrano-Gimenez, A. Perez-Villegas, G. Alvarez, A self-learning anomaly-based web application firewall, in: Computational Intelligence in Security for Information Systems, Springer, 2009, pp. 85-92.

[16] M. Auxilia, D. Tamilselvan, Anomaly detection using negative security model in web application, in: Computer Information Systems and Industrial Management Applications (CISIM), 2010 International Conference on, IEEE, 2010, pp. 481-486.

[17] H. Kapodistria, S. Mitropoulos, C. Douligeris, An advanced web attack detection and prevention tool, Information Management & Computer Security, 19 (2011) 280-299.

[18] A. Tekerek, C. Gemci, O.F. Bay, Development of a hybrid web application firewall to prevent web based attacks, in: Application of Information and Communication Technologies (AICT), 2014 IEEE 8th International Conference on, IEEE, 2014, pp. 1-4.

[19] Y. Huang, A.K. Ghosh, Introducing Diversity and Uncertainty to Create Moving Attack Surfaces for Web Services, Springer New York, 2011.

[20] M. Taguinod, A. Doupé, Z. Zhao, G.J. Ahn, Toward a Moving Target Defense for Web Applications, in: IEEE International Conference on Information Reuse and Integration, 2015, pp. 510-517.

[21] B.-t. Chu, J. Portner, K. Joel, A.-S. Ehab, Moving target defense against cross-site scripting, in, Google Patents, 2016.

[22] S. Sengupta, S.G. Vadlamudi, S. Kambhampati, A. Doupé, Z. Zhao, M. Taguinod, G.-J. Ahn, A Game Theoretic Approach to Strategy Generation for Moving Target Defense in Web Applications, in, AAMAS, 2017.

[23] C. Bailiff, Encoding of universal resource locators in a security gateway to enable manipulation by active content, in, US Patent 20,030,037,232, 2003.

[24] R.E. Salz, E. Kuznetsov, C.A. Dolph, W.T. Lynch, A.M. French, System and method remapping identifiers to secure files, in, Google Patents, 2010.

[25] J. Newsome, D. Song, Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software, (2005).

[26] O. Tripp, M. Pistoia, S.J. Fink, M. Sridharan, O. Weisman, TAJ: effective taint analysis of web applications, in: ACM Sigplan Notices, ACM, 2009, pp. 87-97.

[27] Dewhurst, R. (2012). Damn Vulnerable Web Application (DVWA).

[28] C. Goldberg, Pylot–Open Source Web Performance Tool, in, 2009.

1. In this paper, the virtual URL and the shifting URL are defined as the URL generated by a URL shifting method. [↑](#footnote-ref-1)
2. The user's virtual identity is determined by the virtual cookie and the user source IP address. [↑](#footnote-ref-2)