DRacv:Detecting and Auto-repairing Vulnerabilities in Role-based Access Control in Web Application★

Ke Xu[a], Bing Zhang[a,*], Jingyue Li[b], Haitao He[a], Rong Ren[a] and Jiadong Ren[a,c]

[a]School of Information Science and Engineering, Yanshan University, No. 438, West Section of Hebei Street, Qinhuangdao, 06600, Hebei, China
[b]the Department of Computer science, Norwegian University of Science and Technology, Trondheim 7030, Norway
[c]The Key Laboratory for Software Engineering of Hebei Province, Yanshan University, Qinhuangdao, 06600, Hebei, China

ARTICLE INFO

Keywords:
Web Application Security
Role-based Access Control
Access Control Vulnerability
Vulnerability Detection
Vulnerability Repair

ABSTRACT

Traditional methods for analyzing Broken Access Control (BAC) vulnerabilities have limitations regarding low coverage of access control rules, high false positive rate (FPR), and low detection efficiency. Additionally, state-of-the-art strategies for repairing BAC vulnerabilities utilizing statement-level replacement as a repair method may introduce new logical errors. To address these challenges, we propose a novel approach called DRacv (Detect and Repair Access Control Vulnerabilities) to identify and auto-repair vulnerabilities in Role-Based Access Control (RBAC) mode used in web applications. To detect vulnerabilities, DRacv first constructs a Fine-grained Global Multi-attribute Architectural Navigation Graph model (*FG-MANG*) for web applications through dynamic execution and static analysis, which characterizes full relationships between roles, privileges, and accessible page resources. Based on access control rules extracted from *FG-MANG*, DRacv generates targeted attack payloads to detect BAC vulnerabilities, significantly reducing FPR and eliminating redundant attack payloads. To auto-repair the identified vulnerabilities, DRacv first precisely extracts access control privilege parameters, validation functions, and contextual statements to construct the patch code templates. These templates generate user- and role-level verification patch codes for different users and roles. Instead of changing the vulnerable code, the patch codes behave like firewalls. They are added as separate files and invoked by the web page with vulnerability to defend against access control compromises. DRacv was evaluated on 12 popular open-source web applications in PHP and JAVA. From the applications, DRacv identified 35 vulnerabilities (11 were new) with only one false positive, achieving an FPR of 2.78%. We also compared DRacv's detection results with state-of-the-art studies. Results show that DRacv outperforms those studies regarding the number of vulnerabilities detected and FPR. Among the 35 vulnerabilities detected, DRacv automatically repaired 34 of them, achieving a repair rate of 97.14%. The evaluation results also show that DRacv auto-fixed 18% and 70% more vulnerabilities than the two state-of-the-art auto-repairing methods, respectively.

1. Appendix

1.1. Details of detected vulnerabilities

Table 1 provides detailed information on BAC vulnerabilities, including the locations of the vulnerable code, the *CVE_ID* [1] of the known vulnerabilities, and the reasons for vulnerabilities.

1.2. Patch codes of other nine open-source Web applications

Phpns is an open-source news application that incorporates privilege parameters, namely *username_* and *password_*, to facilitate strict user verification. Additionally, the web application employs the global parameter *rank* for role privilege validation. If the value of *rank* is 0, the application redirects the user to the initial login page. Similar repair templates to this application include AWCMs, Phpoll, Bwapp, JsForum, and OnlineStore.

Mybloggie is a blogging application that offers features to add, delete, and update blog posts. To ensure strict user verification, the application incorporates the validation function *verifyuser()* to address any vulnerabilities present in the *VP index.php*. Furthermore, the role's privilege verification is implemented using the privilege parameter *level*, effectively repairing vulnerabilities on both the *VPs index.php* and *install.php*. Similar repair templates to this application include scarf, Events lister, and Mybb.

Wackopicko is an application with intentionally injected vulnerabilities. The application incorporates two validation functions, namely *require_login()* and *require_admin_login()*, for the strict user verification and the role's privilege verification, respectively. These functions are utilized to address vulnerabilities present in the *VP*, specifically *users/view.php* and *pictures/view.php*. Both functions are nested within the *is_logged_in()* function, which includes the privilege parameter *userid*. Similar repair templates to this application include DVWA.

AWCMs is a web resource management application that encompasses categories such as videos, themes, sounds, and photos. Strict user verification can be enforced by using the

★

*Corresponding author
xuke_kara@163.com (K. Xu); bingzhang@ysu.edu.cn (B. Zhang); jingyue.li@ntnu.no (J. Li); haitao@ysu.edu.cn (H. He); renrongysu@hotmail.com (R. Ren); jdren@ysu.edu.cn (J. Ren)
ORCID(s): https://orcid.org/0009-0009-9740-9454 (K. Xu); https://orcid.org/0000-0002-9867-8439 (B. Zhang); https://orcid.org/0000-0002-7958-391X (J. Li); https://orcid.org/0000-0001-9338-5605 (H. He); https://orcid.org/0000-0002-2245-9133 (J. Ren)

Table 1
The details of detected vulnerabilities

Type	App.	The URL of the VP and input parameters that trigger the vulerabilities	Known with CVE & reported by others without CVE / new	The missing parameters or functions
VPE	Phpns	"user.php?do=loginrec&action=delall"	CVE-2008-6546	*$globalvars['rank'][#]*
		"/manage.php?v=admin"	CVE-2008-6546	
		"/manage.php?v=all"	CVE-2008-6546	
		"/manage.php?do=deleteitems"	CVE-2008-6546	
		"/manage.php?do=search"	CVE-2008-6546	
	AWCMs	"m_cp_avatar.php?KeepThis=true&TB_iframe=true &height=400&width=50"	new	*$_SESSION['awcm_cp']*
		"member.php?id=1"	new	
		"/install/index.php"	new	
		"/control/db_backup.php"	CVE-2010-1066	
	PHPOLL	"modifica_votanti.php"	Reported in [3, 5, 6]	*$COOKIE["$string_cook_login"]* *$COOKIE["$string_cook_password"]*
		"modifica_band.php"	Reported in [3, 5]	
		"modifica_configurazione.php"	Reported in [3, 5]	
	Bwapp	"/backdoor.php"	new	*$_COOKIE["security_level"]*
		"bwapp/install.php"	new	
	DVWA	"/setup.php"	new	*$_COOKIE['security']*
	Scarf	"generaloptions.php"	CVE-2006-5909	*is_admin()*
		"comments.php"	Reported in [2]	
		"showsessions.php"	new	
	Events Lister	"/admin/user_add.php"	CVE-2009-3168	*checkUser()*
		"/admin/setup.php"	Reported in [3, 5]	
		"/admin/add_user.php"	Reported in [4]	
	MyBloggie	"/blog.php/index.php?mode=editcom&post_id=1 &comment_id=2"	CVE-2007-3650	*$_SESSION['level']*
		"install.php"	CVE-2007-3650	
	Mybb	"memberlist.php"	CVE-2018-1000503	*user_permissions()*
		"admin/index.php"	new	
	wackopicko	"/pictures/view.php?picid=10"	Reported in [2, 4]	*require_admin_login()*
	JsForum	"http://localhost:8080/forum/index.jsp? page=message&forum_id=1&thread_id=3&start=0"	Reported in [4]	*sessionType*
		"http://localhost:8080/forum/./index.jsp? page=editmessage&forum_id=0&thread_id=2&reply_id=1&start=0"	Reported in [4]	
	OnlineStore	"http://localhost:8080/onlineStore/order? method=findMyOrdersByPage&pageNumber=2"	new	*user*
		"http://localhost:8080/onlineStore/order? method=getProductById&orderId=202624908720230724200614"	new	
HPE	Phpns	"/manage.php?v=user1"	CVE-2008-6546	*$data['username_']* *$data['password_']*
		"/?sort=article_title/asc" "/?sort=artical_cat/asc" "/?sort=timestamp/asc" "/?sort=artical_author/asc" "/?sort=sort=active/asc"	CVE-2008-6546	
	AWCMs	"member.php?id=2/3"	new	*$username*
	MyBloggie	"/blog.php/index.php?mode=editcom&post_id=1 &comment_id=1/"	CVE-2007-3650	*verifyuser()*
	Wackopicko	"/users/view.php?userid=2/"	Reported in [2, 4]	*require_login()*

privilege parameter *username* to repair BAC vulnerabilities in *VPs*, such as *member.php*. Additionally, the role's privilege verification can be employed to repair BAC vulnerabilities in *VPs* including *m_cp_avatar.php, member.php, install/index.php*, and *db_backup.php* by utilizing the privilege parameter *awcm_cp*. If the value of the *awcm_cp* parameter is set to "*no*", the application will redirect the user to the *index.php* page. As shown in Fig.4.

PHPOLL is a simple voting web application. Its privilege parameters are extracted as *string_cook_login* and *string_cook_password*. They are used in the *VP modifica_votanti.php, modifica_band, and modifica_configurazione* to repair the vulnerability. As shown in Fig. 5.

Bwapp is a vulnerability demonstration platform that utilizes privilege parameters, such as *username* and *password*, to ensure strict user verification and address vulnerabilities in the *VPs backdoor.php* and *install.php*. Additionally, the platform employs the privilege parameter *security_level* to implement security level distinction, namely, role's privilege verification, on these pages. The *security_level* parameter has three possible values: 0, 1, and 2. Its assignment depends on whether the application calls the parameter and if the user's current page level matches the assigned value of the page (*val*). If there is no match, the level defaults to 2. As shown in Fig. 6.

DVWA is a web application specifically designed to identify security vulnerabilities. In order to enforce strict user verification, the application utilizes the validation function *dvwaIsLoggedIn()* to address any vulnerabilities present in the *VP setup.php*. Additionally, the application incorporates the privilege parameter *security* to establish different levels of security on the *VP setup.php*. The *security* parameter consists of four possible values: *low, medium, high*, and *impossible*. The assigned security level is determined based on whether the application calls the *security* parameter and if the user's current page level matches the assigned value (*val*) of the page. If there is no match, the highest security level, *high*, is set as the default. As shown in Fig. 7.

Scarf is a research forum and conference hosted by Stanford. Within this forum, two validation functions, namely *is_admin()* and *require_loggedin()*, are utilized, both of which use *privilege* as the privilege parameter. The purpose of the *require_loggedin()* function is used for strict user verification. On the other hand, the *is_admin()* function is used to enforce the role's privilege verification, addressing vulnerabilities in the *VPs*: *generaloptions.php, comments.php*, and *showsessions.php*. As shown in Fig.8.

Events lister is a PHP application developed for managing activities, meetings, schedules, and other events. In order to mitigate vulnerabilities in the *VPs*, specifically *user_add.php, add_user.php*, and *setup.php*, the application employs the validation function *checkuser()*. This function relies on the privilege parameter $\$\_SESSION['validUser']$. As shown in Fig. 9.

Mybb is an open-source forum platform freely available to the public that adheres to standard forum structure and model. To ensure strict user verification, the application utilizes the validation function *user_permissions()* in conjunction with privilege parameters *uid* and *password*. Additionally, the role's privilege verification is achieved through the validation function *user_admin_permissions()*. Finally, the vulnerabilities in *VP* such as *memberlist.php* and *admin/index.php* are repaired. As shown in Fig. 10.

OnlineStore is an e-commerce platform developed in JAVA and constructed using the MVC framework. The application integrates strict user verification and the role's privilege verification by utilizing the privilege parameter *user*. This parameter effectively repairs vulnerabilities in the *VPs* such as *order?method=findMyOrdersByPage* and *order?method=getProductById*. As shown in Fig. 11.

JsForum is a comprehensive J2EE forum, also known as a bulletin board, developed using the Struts MVC framework. The application emphasizes strict user verification by incorporating privilege parameters such as *sessionUsername* and *sessionPassword*. Additionally, the role's privilege verification is achieved through the utilization of the *sessionType* parameter, which is employed to address and resolve vulnerabilities present in the *VPs*, namely *message.jsp* and *editmessage.jsp*. As shown in Fig. 12.

user.php (before patch)

```
<?php
include("inc/header.php");
$do = $_GET['do'];
if (!$do) then {
    $globalvars['page_name'] = 'user management';
    ...
?>
```

user.php (after patch)

```
<?php
include("patch.php");
repair();
include("inc/header.php");
$do = $_GET['do'];
if (!$do) then {
    $globalvars['page_name'] = 'user management';
    ...
?>
```

patch.php (It is where the patch code exists.)

```
<?php
// To encapsulate the patch code in a function style
function repair(){
// Implement strict user verification by employing the privilege parameters $data["username"] and $data["password"] to mitigate HPE vulnerabilities, as detailed in section 3.3.1.1. Ensure that these parameters are set before proceeding.
if (!(isset($data["username"]) && isset($data["password"]))) then
    {header("Location: index.php?do=privilegedenied");}
// Implement role's privilege verification by employing the privilege parameter $globalvars["rank"] to mitigate VPE vulnerabilities. Ensure that these parameters are set before proceeding.
if  (isset($globalvars["rank"])) then{
// If $globalvars['rank'] equals 0, it indicates the absence of privilege, and access should be denied.
    if ($globalvars["rank"] == 0) then
        {header("Location: index.php?do=privilegedenied");}
}
?>
```

Figure 1: Access control patch code for *Phpns* in PHP using privilege parameters.

index.php (before patch)

```
<?php
define('IN_MYBLOGGIE', true);
session_start();
header("Cache-control: private");
ini_set("magic_quotes_runtime",0);
$mybloggie_root_path = './';
...
?>
```

index.php (after patch)

```
<?php
include("patch.php");
repair();
define('IN_MYBLOGGIE', true);
session_start();
header("Cache-control: private");
ini_set("magic_quotes_runtime",0);
$mybloggie_root_path = './';
...
?>
```

patch.php (It is where the patch code exists.)

```
<?php
function repair(){
// Implement strict user verification by calling the location of the validation function exists, as detailed in section 3.3.1.1.
include('includes/function.php');
// Employ the validation function Verifyuser() to mitigate HPE vulnerabilities.
Verifyuser();
// Implement role's privilege verification by employing the privilege parameter $_SESSION['level'] to mitigate VPE vulnerabilities.
Ensure that these parameters are set before proceeding.
if (isset($_SESSION['level'])) then{
// Check the following code which are obtained in secion 3.3.1.2). This means the absence of privilege, and access should be denied.
    if (getpage($url) is not in $_SESSION['level'].Gr) then{ header('location:./login.php');}}
}
?>
```

Figure 2: Access control patch code for Mybloggie application in PHP using validation functions.

pictures/view.php (before patch)

```
<?php
require_once("../include/html_functions.php");
require_once("../include/functions.php");
if (!session_id()) then{
    session_start();
    }
require_login();
$no_pic = False;
...
?>
```

pictures/view.php (after patch)

```
<?php
require_once("patch.php");
repair();
require_once("../include/html_functions.php");
require_once("../include/functions.php");
if (!session_id()) then{
    session_start();
    }
require_login();
$no_pic = False;
...
?>
```

patch.php (It is where the patch code exists)

```
<?php
function repair(){
// Call the location of the validation function exists, as detailed in section 3.3.1.1.
include 'functions.php';
// Implement strict user verification by employing the validation function require_login() to mitigate HPE vulnerabilities.
require_login();
// Implement role's privilege verification by employing the validation function reauire_admin_login() to mitigate VPE vulnerabilities.
reauire_admin_login();
}
?>
```

Figure 3: Access control patch code for Wackopicko application in PHP using privilege parameters and validation functions.

member.php (before patch)

```
<?php
$page = ’member’;
include ("header.php");
include ("includes/window_top.php");
$gid = $_GET[’id’];
?>
<title><?php print $title; ?></title>
. . .
```

member.php (after patch)

```
<?php
include("patch.php");
repair();
$page = ’member’;
include ("header.php");
include ("includes/window_top.php");
$gid = $_GET[’id’];
?>
<title><?php print $title; ?></title>
. . .
```

patch.php (containing the patch code)

```
<?php
function repair (){
// Implement strict user verification by employing the privilege parameters $_SESSION[’username’]) to mitigate HPE vulnerabilities, as detailed in section 3.3.1.1. Ensure that these parameters are set before proceeding.
if (! isset($_SESSION[’username’])) then{
    header(’location:./login.php’);}
// Implement role’s privilege verification by employing the privilege parameter $_SESSION[’awcm_cp’]) to mitigate VPE vulnerabilities. Ensure that these parameters are set before proceeding.
if (isset($_SESSION[’awcm_cp’])) then{
    if ($_SESSION[’awcm_cp’] == ’no’) then
        {header("location:./index.php");}
}}
?>
```

Figure 4: Access control patch code for AWCMs application in PHP using privilege parameters.

modifica_votanti.php (before patch)

```
<html>
<head>
<title>PHPOLL</title>
<link rel="stylesheet" href="../css/phpoll_layout.css" title="phpoll layout" />
</head>
</html>
<?php
include "../config/config.php";
...
?>
```

modifica_votanti.php (after patch)

```
<html>
<head>
<title>PHPOLL</title>
<link rel="stylesheet" href="../css/phpoll_layout.css" title="phpoll layout" />
</head>
</html>
<?php
include("patch.php");
repair();
include "../config/config.php";
...
?>
```

patch.php (containing the patch code)

```
<?php
function repair (){
// Implement strict user verification by employing the privilege parameters $_COOKIE[$string_cook_login] to mitigate HPE vulnerabilities, and implement role’s privilege verification by employing the privilege parameter $_COOKIE[$string_cook_password] to mitigate VPE vulnerabilities. As detailed in section 3.3.1.1. Ensure that these parameters are set before proceeding.
if (!(isset($_COOKIE[$string_cook_login]) && isset($_COOKIE[$string_cook_password]))) then{
    header(’location:./index’);
}}
?>
```

Figure 5: Access control patch code for Phpoll application in PHP using privilege parameters.

backdoor.php (before patch)

```
<?php
if (isset($_REQUEST["upload"])) then{
    $dir=$_REQUEST["uploadDir"];}
    if (phpversion()<'4.1.0') then{
        $file=$HTTP_POST_FILES["file"]["name"];}
@move_uploaded_file($HTTP_POST_FILES["file"]["tmp_name"]);
...
?>
```

backdoor.php (after patch)

```
<?php
include("patch.php");
repair();
if (isset($_REQUEST["upload"])) then{
    $dir=$_REQUEST["uploadDir"];}
    if (phpversion()<'4.1.0') then{
        $file=$HTTP_POST_FILES["file"]["name"];}
@move_uploaded_file($HTTP_POST_FILES["file"]["tmp_name"]);
...
?>
```

patch.php (containing the patch code)

```
<?php
function repair (){
// Implement strict user verification by employing the privilege parameters $username and $password to mitigate HPE vulnerabilities,
as detailed in section 3.3.1.1. Ensure that these parameters are set before proceeding.
if (!(isset($username) && isset($password))) then{
    header('location:./index');}
// Implement role's privilege verification by employing the privilege parameter $_COOKIE['security_level']to mitigate VPE vulnera-
bilities. Ensure that this parameters are set before proceeding.
if (isset($_COOKIE['security_level'])) then{
    if ($_COOKIE['security_level'] != get_current_user($user).level) then{
        reset('security_level','2'); }
}}
?>
```

Figure 6: Access control patch code for Bwapp application in PHP using privilege parameters.

setup.php (before patch)

```
<?php
define( 'DVWA_WEB_PAGE_TO_ROOT', '' );
require_once DVWA_WEB_PAGE_TO_ROOT . 'dvwa/includes/dvwaPage.inc.php';
dvwaPageStartup( array( 'phpids' ) );
$page = dvwaPageNewGrab();
$page[ 'title' ] = 'Setup' . $page[ 'title_separator' ].$page[ 'title' ];
$page[ 'page_id' ] = 'setup';
...
?>
```

setup.php (after patch)

```
<?php
require_once 'patch.php';
repair();
define( 'DVWA_WEB_PAGE_TO_ROOT', '' );
require_once DVWA_WEB_PAGE_TO_ROOT . 'dvwa/includes/dvwaPage.inc.php';
dvwaPageStartup( array( 'phpids' ) );
$page = dvwaPageNewGrab();
$page[ 'title' ] = 'Setup' . $page[ 'title_separator' ].$page[ 'title' ];
$page[ 'page_id' ] = 'setup';
...
?>
```

patch.php (containing the patch code)

```
<?php
function repair (){
// Implement strict user verification by calling the location of the validation function exists, as detailed in section 3.3.1.1.
include('includes/function.php');
// Employ the validation function dvwaIsLoggedIn() to mitigate HPE vulnerabilities.
dvwaIsLoggedIn();
// Implement role's privilege verification by employing the privilege parameter $_COOKIE['security'] to mitigate VPE vulnerabilities. Ensure that this parameters are set before proceeding.
if (isset($_COOKIE['security'])) then{
    if ($_COOKIE['security'] != get_current_user($user).level) then{
        reset('security', 'high');
}}
?>
```

Figure 7: Access control patch code for DVWA application in PHP using validation function and privilege parameter.

generaloptions.php (before patch)

```
<?php include_once("header.php");
if (isset($_POST['submit'])) then {
    for foreach ($_POST as $name =>$value) do {
        $name = mysql_real_escape_string($name);
        $name = str_replace("_", " ", $name);
        $value = mysql_real_escape_string($value);
        query("UPDATE options SET value='$value'
WHERE name='$name'");
}}
. . .
?>
```

generaloptions.php (after patch)

```
<?php include_once("header.php");
include_once("patch.php");
repair();
if (isset($_POST['submit'])) then {
    for foreach ($_POST as $name =>$value) do {
        $name = mysql_real_escape_string($name);
        $name = str_replace("_", " ", $name);
        $value = mysql_real_escape_string($value);
        query("UPDATE options SET value='$value'
WHERE name='$name'");
}}
. . .
?>
```

patch.php (containing the patch code)

```
<?php
function repair (){
// Call the location of the validation function exists, as detailed in section 3.3.1.1.
include "...\scarf \\functions.php";
// Implement strict user verification by employing the validation function require_loggedin() to mitigate HPE vulnerabilities.
require_loggedin();
// Implement role's privilege verification by employing the validation function is_admin() to mitigate VPE vulnerabilities.
is_admin();
?>
```

Figure 8: Access control patch code for Scarf application in PHP using validation functions.

user_add.php (before patch)

```
<?php
?>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Add a user</title>
</head>
<body>
<h1>Add User</h1>
...
</body>
</html>
```

user_add.php (after patch)

```
<?php
include("patch.php");
repair();
?>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Add a user</title>
</head>
<body>
<h1>Add User</h1>
...
</body>
</html>
```

patch.php (containing the patch code)

```
<?php
function repair () {
// Call the location of the validation function exists, as detailed in section 3.3.1.1.
include('functions.php');
// Implement strict user verification and role's privilege verification by employing the validation function checkuser() to mitigate HPE and VPE vulnerabilities.
checkuser();
?>
```

Figure 9: Access control patch code for Events lister application in PHP using validation function.

memberlist.php (before patch)

```
<?php
define("IN_MYBB", 1);
define("IGNORE_CLEAN_VARS", "sid");
define('THIS_SCRIPT', 'member.php');
define("ALLOWABLE_PAGE","register,do_register,login,
do_login,logout,lostpw,do_lostpw,activate,resendactivation,
do_resendactivation,resetpassword,viewnotes");
$nosession['avatar'] = 1;
...
?>
```

memberlist.php (after patch)

```
<?php
include("patch.php");
repair();
define("IN_MYBB", 1);
define("IGNORE_CLEAN_VARS", "sid");
define('THIS_SCRIPT', 'member.php');
define("ALLOWABLE_PAGE","register,do_register,login,
do_login,logout,lostpw,do_lostpw,activate, resendactivation,
do_resendactivation,resetpassword,viewnotes");
$nosession['avatar'] = 1;
...
?>
```

patch.php (containing the patch code)

```
<?php
function repair(){
// Call the location of the validation function exists, as detailed in section 3.3.1.1.
include 'Upload/inc/functions.php';
// Implement strict user verification by employing the validation function validate_password_from_uid() to mitigate HPE vulnerabilities.
validate_password_from_uid();
// Implement role's privilege verification by employing the validation function user_admin_privilege() to mitigate VPE vulnerabilities.
user_admin_privilege();
}
?>
```

Figure 10: Access control patch code for Mybb application in PHP using validation functions.

OrderServlet (before patch)

```
public String getProductById(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
try {
String sOrderId = request.getParameter("orderId");
Order order = this.orderService.getOrderById(sOrderId);
request.setAttribute("order", order);
return "/jsp/order_info.jsp";
} catch (Exception var5) {
var5.printStackTrace();
request.setAttribute("sMessage", "defeat");
return "/jsp/message.jsp";
}}
```

OrderServlet (after patch)

```
public String getProductById(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
try {
patch obj = new patch();
obj.repair();
String sOrderId = request.getParameter("orderId");
Order order = this.orderService.getOrderById(sOrderId);
request.setAttribute("order", order);
return "/jsp/order_info.jsp";
} catch (Exception var5) {
var5.printStackTrace();
request.setAttribute("sMessage", "defeat");
return "/jsp/message.jsp";
}}
```

patch.php (containing the patch code)

```
public class patch {
public String String repair(){
// Implement strict user verification and role's privilege verification by employing the privilege parameters "user" to mitigate HPE and VPE vulnerabilities, as detailed in section 3.3.1.1. Ensure that these parameters are set before proceeding.
User user = (User)request.getSession().getAttribute("user");
if (!user) then{
    request.setAttribute("sMessage", "login again");
    return "/jsp/message.jsp";}
}}
```

Figure 11: Access control patch code for OnlineStore application in JAVA using privilege parameters.

editmessage.jsp (before patch)

```
<% String requestPage = request.getParameter("page"); %>
<%
if (requestPage == null) then{
    requestPage = "0";
    }%>
<%
if (requestPage.equals("thread")) then{ %>
    <% String forum_id = request.getParameter("forum_id"); %>
    <jsp:include page="thread.jsp" flush="true" >
    <jsp:param name="forum_id" value="<%= forum_id %>" />
    </jsp:include>
...
}%>
```

editmessage.jsp (after patch)

```
<%@ page import="patch" %>
<% repair();%>
<% String requestPage = request.getParameter("page"); %>
<%
if (requestPage == null) then{
    requestPage = "0";
    }%>
<%
if (requestPage.equals("thread")) then{ %>
    <% String forum_id = request.getParameter("forum_id"); %>
    <jsp:include page="thread.jsp" flush="true" >
    <jsp:param name="forum_id" value="<%= forum_id %>" />
    </jsp:include>
...
}%>
```

patch.php (containing the patch code)

```
<%
String repair() {
// Implement strict user verification by employing the privilege parameters "sessionUsername" and "sessionPassword" to mitigate HPE vulnerabilities, as detailed in section 3.3.1.1. Ensure that these parameters are set before proceeding.
if (!(session.getAttribute("sessionUsername") != null && session.getAttribute("sessionPassword") != null)) then{
    <a href="./register.jsp">Register</a>
    }
// Implement role's privilege verification by employing the privilege parameter "sessionType" to mitigate VPE vulnerabilities. Ensure that these parameters are set before proceeding.
if (isset(session.getAttribute("sessionType")) then{
    <% session.getAttribute("sessionType").equals("Admin")){ %>
    <jsp:include page="./include/table_title.jsp" flush="true">
    <jsp:param name="title" value="index" />
    <jsp:param name="colspan" value="1" />
    <jsp:param name="align" value="middle" />
    </jsp:include>
    <% } %>
    }
}%>
```

Figure 12: Access control patch code for JsForum application in JAVA using privilege parameters.

References

[1] CVE, 2022. Access. http://cve.mitre.org/.

[2] Deepa, G., Thilagam, P.S., Praseed, A., Pais, A.R., 2018. Detlogic: A black-box approach for detecting logic vulnerabilities in web applications. Journal of Network and Computer Applications 109, 89–109.

[3] Gauthier, F., Merlo, E., 2012. Fast detection of access control vulnerabilities in php applications, in: 2012 19th Working Conference on Reverse Engineering, IEEE. pp. 247–256.

[4] Li, X., Si, X., Xue, Y., 2014. Automated black-box detection of access control vulnerabilities in web applications, in: Proceedings of the 4th ACM Conference on Data and Application Security and Privacy, pp. 49–60.

[5] Sun, F., Xu, L., Su, Z., 2011. Static detection of access control vulnerabilities in web applications, in: USENIX Security Symposium.

[6] Xia, Z., Peng, G., Hu, H., 2018. A test case generation approach for exploiting access control vulnerabilitiesbased on policy inference(in chinese). Computer Engineering and Applications 54, 63–68.