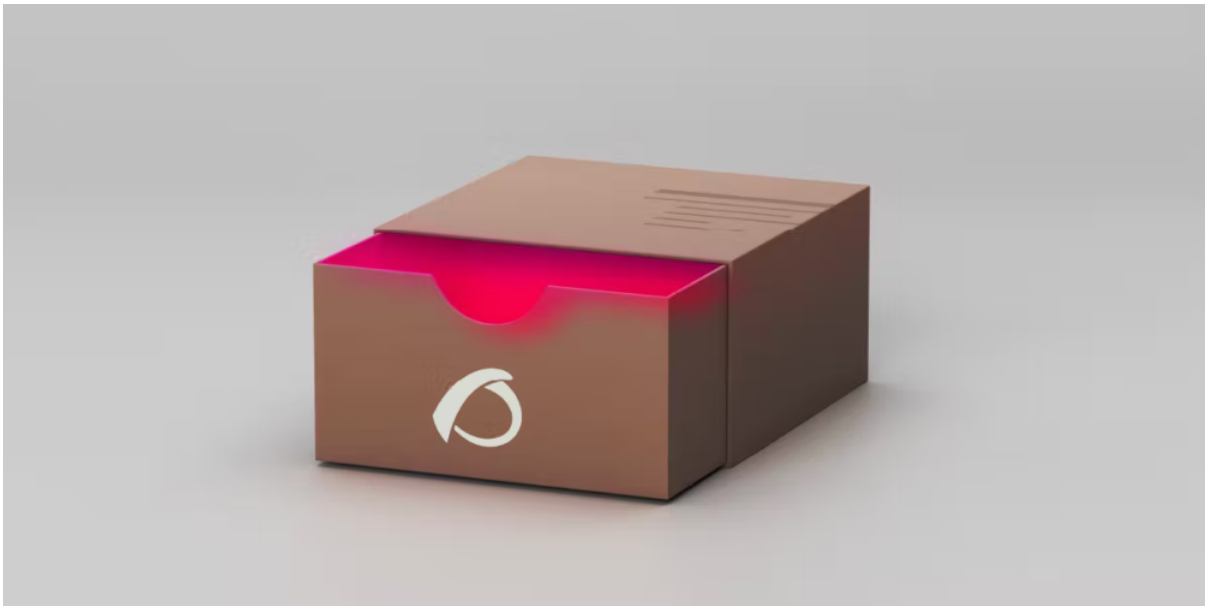
Pandora FMS 742: Critical Code Vulnerabilities Explained

BY DENNIS BRINKROLF | SEPTEMBER 22, 2020

Security



Pandora FMS is an open source software for monitoring IT infrastructure and networks. It can monitor the status and performance of network equipment, operating systems, virtual infrastructure and all different kinds of security-sensitive applications and systems such as firewalls, databases and web servers. Its enterprise edition is used by many industry leaders, for example AON, Allianz and Toshiba.

During our web application security research, we discovered several vulnerabilities in Pandora FMS version 742. These allow remote attackers to execute arbitrary code on any Pandora FMS server. No prior knowledge, access privilege or specific configuration is required by an attacker. The systems that are connected for monitoring to Pandora FMS may be directly prone to further attacks. We reported all issues responsibly to the affected vendor who released a security patch version 743 immediately.

In this blog post we analyze the technical root cause of the most critical vulnerability and how attackers could have exploited it.

Impact

During the analysis of Pandora FMS 742 console we found the following code vulnerabilities:

- SQL Injection (pre authentication) (CVE-2021-32099)
- Phar deserialization (pre authentication) (CVE-2021-32098)
- Remote File Inclusion (lowest privileged user) (CVE-2021-32100)
- Cross-Site Request Forgery (CSRF)

Our focus is on a severe SQL injection vulnerability. It can be remotely exploited without any access privileges and enables an attacker to completely bypass the administrator authentication. This enables in the end to execute arbitrary code on the system.

Pandora FMS is mostly used in internal networks and is typically not directly accessible to a remote attacker. However, the SQL injection can be exploited via a Cross-Site Request Forgery attack. A single person whose browser can reach the Pandora FMS installation and who is visiting a maliciously prepared website would be sufficient to carry out the attack and to take over the entire server. The targeted person does not need to have an account nor any privileges in Pandora FMS. During our analysis we also found several Pandora instances that are directly accessible via the internet.

Monitoring solutions are attractive targets for attackers, as these typically have access to the devices that they monitor, and are a starting point to compromise other parts of the infrastructure. For demonstration purposes we've created a short video that shows how quick and easy a server is compromised.

Technical Analysis

In the following, we will look at the root cause of the vulnerability in the source code of Pandora FMS, written in PHP. For this purpose we will first introduce the security mechanisms used by Pandora FMS to sanitize user controlled inputs and highlight potential problems. Finally, we will see how this led to a critical vulnerability that enables an authentication bypass.

Security Mechanism

In Pandora FMS' source code, user input is typically sanitized with the help of a custom function called `io_safe_input()`. It sanitizes string values by using the PHP built-in function `htmlspecialchars()` which encodes certain HTML markup characters (`"<>'`). Additionally, other security measures are taken in this function.

/include/functions_io.php

```
 72 function io_safe_input($value) {
  ⋮
 94     $valueHtmlEncode = htmlentities($value, ENT_QUOTES, 'UTF-8', true);
  ⋮
128     return $valueHtmlEncode;
129 }
```

The developers of Pandora FMS also implemented a wrapper function for retrieving GET and POST parameters called `get_parameter()`. This function uses the function `io_safe_input()` as described above to sanitize user input that is retrieved from `$_GET` or `$_POST` parameters. Such a wrapper function is often used to avoid that the developers have to worry about cleaning up the values. A wrapper function is definitely useful, and yet there are some pitfalls to be aware of.

1. The usage of this function is optional. It is still possible to access the `$_GET` or `$_POST` variables directly which are not sanitized. The direct access occurs several times within the Pandora FMS code base and this has led to security issues in the past.
2. The sanitized data retrieved by the wrapper may still lead to security problems because input has to be sanitized depending on the markup context. For example, the `htmlspecialchars()` function does not protect against Cross-Site Scripting vulnerabilities if user input is embedded into various JavaScript code parts. Developers may blindly trust the wrapper function to be secure without knowing what it actually does.
3. There are many possibilities besides GET and POST parameters to process user input, e.g. cookies or HTTP headers. However, all possible user inputs should always be sanitized.

Unauthenticated SQL Injection (CVE-2021-32099)

Let's have a look at how user input is processed in the Chart Generator of Pandora FMS. When accessing the Chart Generator, first the authentication is checked.

/include/chart_generator.php

```
71  // Try to initialize session using existing php session id.
72  $user = new PandoraFMS\User(['phpsessionid' => $_REQUEST['session_id']]);
73  if (check_login(false) === false) {
74      // Error handler.
 ⋮
96  }
97
98  // Access granted.
```

As we can see in line 72 of *chart_generator.php*, the user input is fetched from the `$_REQUEST` superglobal which contains GET and POST parameters, as well as cookie values. The latter is probably the reason why `get_parameter()` was not used here. The user input `$_REQUEST['session_id']` is passed to the constructor of the class `PandoraFMS\User` without any sanitization. Then, the function `check_login()` is used to check if a login session variable is set and valid. All in all, the function `check_login()` evaluates as *true* if a user with the given session ID exists and then the access is granted.

The following snippet shows what happens in the constructor of class `PandoraFMS\User` with the attacker controlled value `$data['phpsessionid']`.

/include/lib/User.php

```
73              ['id_session' => $data['phpsessionid']]
74          );
75
76          if ($info !== false) {
77              // Process.
78              $session_data = session_decode($info['data']);
79              $this->idUser = $_SESSION['id_usuario'];
80
81              // Valid session.
82              return $this;
83          }
```

In line 73, the user controlled parameter is passed to the function `db_get_row_filter()`. This function uses a couple of internal functions that dynamically builds a SQL query based on the provided table name and a condition supplied as an array. At this point, it concatenates the attacker controlled variable directly into a SQL `WHERE` clause without proper sanitization which leads to a SQL Injection (line 762 in *mysql.php*).

/include/lib/mysql.php

```
848  function db_get_row_filter($table, $filter, $fields=false)
849  {
 ⋮
861      $filter = db_format_array_where_clause_sql($filter, ' WHERE ');
 ⋮
868      $sql = sprintf('SELECT %s FROM %s %s', $fields, $table, $filter);
```

/include/lib/mysql.php

```
660  function db_format_array_where_clause_sql($values, $prefix=false)
661  {
668      $query = '';
 ⋮
709      foreach ($values as $field => $value) {
 ⋮
762          $query .= sprintf("%s = '%s'", $field, $value);
 ⋮
771      }
772
773      return (!empty($query) ? $prefix : '') . $query;
```

The SQL injection allows an attacker to malform the constructed SQL and, thus, the result set of the database query. From here, an attacker can control the data in `$info['data']` in line 71 of *User.php*. The PHP function `session_decode()` is then used to load session data from `$info['data']` and to populate it into the current `$_SESSION` in line 78. This way, any user can be impersonated including an administrator with full access privileges by loading its user ID. As a result, the SQL Injection can be used to authenticate as any user. Due to the criticality of the vulnerability we are omitting the exact exploitation details at this point.

Note that the function `session_decode()` is capable of deserializing arbitrary objects similar to the function `unserialize()`. This means that an attacker could deserialize arbitrary objects via the SQL Injection and this can be another attack vector. In the end, a login bypass is sufficient for an attacker because as an administrator there are already possibilities to execute code (also see CVE-2020-13851).

Patch

The vulnerability has been patched by the vendor in the latest version by using the previously introduced wrapper function `io_safe_input()` to sanitize input. This patch is secure for the applied context and, at the same time, it is difficult to verify for other developers as discussed in the previous section.

/include/lib/User.php

```
71          $info = \db_get_row_filter(
72              'tsessions_php',
73              ['id_session' => io_safe_input($data['phpsessionid'])]
74          );
```

By looking at this patch in line 73 we don't know the exact context of the SQL query inside of db_get_row_filter() and if `io_safe_input()` is a sufficient sanitization. In case `db_get_row_filter()` would internally craft a SQL query and embed the user-supplied data without surrounding it by quotes (`'`), the input sanitization designed for HTML markup would not be sufficient because the attackers payload would not need any quotes for exploitation. Adding context-sensitive input sanitization (escaping VS. type casting) or, even better, prepared statements into the database wrapper functions themself would enable a safer usage of these functions independently of the user-supplied filter.

Timeline

Date	What

dor

Summary

In this blog post we analyzed a critical security vulnerability found in Pandora FMS, a popular IT monitoring solution used by big industry leaders. This vulnerability can lead to a complete takeover of the application and put further network systems at risk. We've evaluated its root cause and how different security mechanisms in the code can have pitfalls and lead to such vulnerabilities. Due to the severity of the issues we've postponed this release for several months. If you are hosting Pandora FMS and did not update your installation yet, we highly recommend to do so now. We would like to thank the Pandora FMS team who quickly released a patch version 743 after our reports.

You can join the discussion about this vulnerability in our community forum.

DENNIS BRINKROLF
Security Researcher
in