CVE-2017-2641Moodle 远程代码执行

1、概述

[CVE-2017-2641]Moodle 远程代码执行漏洞。漏洞(CVE-2017-2641)允许攻击者在易受攻击的 Moodle 服务器上执行 PHP 代码。这个漏洞实际上由许多小漏洞组成，如文章中所述。
Moodle 是一个非常受欢迎的学习管理系统，部署在世界各地的许多大学，包括麻省理工学院，斯坦福大学，剑桥大学和牛津大学等顶尖研究所。
Moodle 存储了大量敏感信息，如成绩，测试和学生私人数据，使其成为一个重要的目标，这是我审计它的主要原因。
此漏洞适用于几乎所有部署的 Moodle 版本，如“受影响版本”部分所示。 我建议所有 Moodle 的管理员应用 安全补丁 。
0x02 受影响版本
3.2～3.2.1
3.1～3.1.4
3.0～3.0.8
2.7.0～2.7.18 及其他不受支持的版本
0x03 技术说明第 1 部分-大部分代码存在问题
Moodle 是一个非常大的系统，它包含数千个文件，数百个不同的组件和大约 200 万行的 PHP 代码。 因此，显而易见的是，不同开发人员编写代码的不同部分，这些部分相互影响。
在下面的白皮书中，我将演示如何使用有漏洞的代码，其实主要原因是太多开发人员没有写文档导致的严重逻辑漏洞，几乎所有具有大量代码库的系统都可能会发生逻辑漏洞。
在系统的内置 Ajax 机制中可以看到一个明显的“相同特征，不同代码”的情况。
Moodle 具有动态 Ajax 系统，允许不同的组件使用系统的内置 Ajax 接口。 Moodle 的方式是使用“External Functions ”。使用内置 Ajax 机制的每个组件都注册自己的“ External Functions ”，指定被调用的组件，函数名称和使用它所需的权限。
当组件希望使用 Ajax 接口时，他们可以简单地调用“ service.php ”文件，提供它们之前注册的外部函数的名称。Moodle 的这种方式允许组件开发者使用其内置的 Ajax 界面，这样可以为他们节省自己编写一个新组件的麻烦。
但是，当 Moodle 的核心开发人员开始使用这个接口时，问题就出现了。
如果有一个组件需要通过 Ajax 请求来更改用户的首选项，它就会调用“ setuserpref.php ”文件，指定要更改的属性的名称和值。 这可以在以下代码中看到：

```
// Check access.
if (!confirm_sesskey()) {
    print_error('invalidsesskey');
}

// Get the name of the preference to update, and check it is allowed.
$name = required_param('pref', PARAM_RAW);
if (!isset($USER->ajax_updatable_user_prefs[$name])) {
    print_error('notallowedtoupdateprefremotely');
}

// Get and the value.
$value = required_param('value', $USER->ajax_updatable_user_prefs[$name]);

// Update
if (!set_user_preference($name, $value)) {
    print_error('errorsettinguserpref');
}

echo 'OK';
```

在代码的中间，在突出显示的行中，我们可以看到 Moodle 正在尝试确保需要更改的首选项在“ ajax_updatable_user_prefs ”数组中的定义，该数组定义了哪些首选项可以通过 Ajax 更改。因为 Moodle 不希望像我们这样的恶意攻击者改变任何重要的地方。

虽然大多数用户即使不通过 Ajax 接口，首选项也可以通过其他措施更改，但是 Moodle 的开发人员试图提前预防，防止以后被滥用此机制。

直到添加了外部函数“ update_user_preferences ”。这个函数被添加来替换旧的“ update_users ”函数，这能够“potentially [be] used to update any user attribute”，这显然是一件非常糟糕的事情;由于旧功能仅用于更新用户的首选项，因此显然无需让其更改任何其他内容。

旧功能与新功能之间的主要区别在于，旧功能无法通过 Ajax 界面进行访问，而新的功能可能会因为所谓的危险功能而改变每个用户属性的功能。

最重要的是，他们实施了适当的权限检查，所以即使攻击者可以利用使用用户的优先权限，它只能在自己的用户身上利用它。

但是，这些 preferences 在稍后的“ eval ”或“ exec ”调用中使用并不重要，无论哪个用户正在利用危险的功能，只要它被利用。

让我们先看一下这个函数的代码：

public static function update_user_preferences($userid, …, $preferences) { ... // If we are trying to edit our own user preferences if ($userid == $USER->id) { // Requires the capability to edit our own profile, which we have require_capability('moodle/user:editownmessageprofile', $systemcontext); } else { // Otherwise, we are tring to edit someone else's preferences /* Require admin capabilities... */ } // Set the user's preferences. foreach ($preferences as $preference) { set_user_preference($preference['type'], $preference['value'], $userid); } ...}

通过查看代码，我们可以看到，由于进行了 preferences 检查，我们只能编辑自己的首选项，但是，仍然缺少一些东西。虽然代码确保我们只能编辑我们自己的用户首选项，但它不检查我们改变哪个 preferences ，而与其他负责更改用户首选项的 Ajax 页面相反 – “ setuserpref.php ”。
一个典型的例子，如果不同的开发人员，在不同的时间，不同的需求，为不同的功能写入不同的代码。 这一次，他们假设用户 preferences 不能以任何恶意的方式利用。他们认为这是不可利用的。

2、利用不可利用的

有一个原因，用户 preferences 被认为是不可利用的。它们在系统操作方面几乎没有任何影响 – 它们不能用于 DB 查询，它们不定义任何组件，它们唯一的影响是系统的 GUI 部分，甚至只有 miner 方式。
那么，我们要做什么呢?
好了，我们先看看系统的 GUI 部分如何工作。
Moodle 正在使用 Blocks mechanism 来允许组件向用户显示相关数据。这些块可以由用户添加和删除。
其中的一个块–“ course_overview ”用于向用户显示其已注册课程的列表。为了储存注册的课程顺序，课程概述块机制使用一个名为“ course_overview_course_sortorder ”的特定用户 preference 。此 preference 存储用户注册的所有课程 ID 的列表，并在其注册时排序。此列表使用逗号分隔课程 ID，因此以下代码行用于拆分列表：

```
    return explode(',', $value); // Split the IDs using a
comma
```

但如果 preference 是空的，会发生什么?在这种情况下，块机制尝试检索旧版首选项“ course_overview_course_order ”，其中已不同的方式再次包含所有课程 ID 的列表。
这次，为了检索到 ID，块机制实际上执行：

```
    $order = unserialize($value); // Unserialize the cours
e IDs
```

一个 unserialize 调用。
另一个典型的例子是遗留代码和向后兼容性如果仍然使用它会损害整个系统，以及不同开发人员如何使用完全不同的方法实现完全相同的功能。
对我们来说，这意味着我们现在可以利用 Object 注入攻击。因为 Moodle 是过滤用户输入的，这局限着我们注入的方式：
我们不能注入 Null 字节。在这意味着我们不能设置任何受保护的或私有的 Object 属性，因为当 PHP 序列化它们时，它们将空字节添加到它们的序列化声明中。 虽然在代码库中

有很多类，但大多数是不可达的。当我们的有效载荷未经序列化时，或者使用自动加载功能无法访问时，它们不包括在内。
这些限制导致了一个非常困难的 Object 注入。

3、从 Object 注入中获取乐趣

由于规定的限制，我们只能使用已经包含类的公共属性。我们也不能使用依赖于任何受保护或私有属性的任何代码，因为它们将被初始化为其默认值，或者大部分时间只是一个 NULL。
这缩小了我们的攻击面。几乎所有的类都以某种方式使用受保护的属性，并且它们中大多数没有任何公共属性。
第一步是了解我们可以使用哪些 magic PHP ：
我们可以调用“ __wakeup() ”，当 object 被反序列化时调用“ __destruct() ”，当 object 被销毁时调用 “ __toString() ” 。
在我们的有效负载被非串行化之后看到正在执行的代码，我们将看到我们的非序列化的有效负载被视为一个数组：

```
    function block_course_overview_update_myorder($sortord
er) { // $sortorder is our unserialized payload. $value =
 implode(',', $sortorder); ... set_user_preference('cours
e_overview_course_sortorder', $value);}
```

此函数尝试将我们的非序列化数组成员加入一个大的字符串。但是，如果我们的一个成员实际上是一个 Object ，那么它的“ __toString() ”方法将被调用。
但是我们可以用“ __toString() ”做什么?
我们来看看” attribute_format “抽象类如何实现自己的“ __toString() ”方法。

```
/**
 * Convert this to an element and then to a string
 * @return string
 */
public function __toString() {
    return $this->determine_format()->html();
}
```

看起来十分简单，我们最来看看我们可以访问的一个代码流，通过调用继承自“ attribute_format ”类的“ feedback ”类的“ decision_format() ”方法：

```php
/**
* Create a text_attribute for this ui element.
*
* @return text_attribute
*/
public function determine_format() {
    return new text_attribute(
        $this->get_name(),
        $this->get_value(),
        $this->get_label(),
        $this->is_disabled()
    );
}

/**
* Determine if this input should be disabled based on the other settings
*
* @return boolean Should this input be disabled when the page loads.
*/
public function is_disabled() {
    ...
    if ($this->grade->grade_item->is_overridable_item() …) {
        $overridden = 1;
    }
    …
}
```

看到“ is_overridable_item() ”调用了吗?这是另一种方法调用，但这次使用 object‘s 的属性之一作为 object 。

现在，我们取得了一些进展。因为我们控制属性，我们控制被调用的 object ，就可以扩展我们的攻击面了。 实现“ is_overridable_item() ”方法的类之一是“ grade_item ”类。

经过一系列方法调用，我们最终到达“update”方法。

```php
/**
 * Is the grade item overridable
*/
public function is_overridable_item() {
    ...
    return ... ($this->is_external_item() or $this->is_calculated() ...);
}
```

```
/**
 * Checks if grade calculated. Returns this object's calculation.
*/
public function is_calculated() {
    ...
    if (!$this->calculation_normalized and strpos($this->calculation, '[[') !== false) {
        $this->set_calculation($this->calculation);
    }
    ...
}
```

```
/**
 * Sets this item's calculation (creates it) if not yet set, or
 * updates it if already set (in the DB). If no calculation is given,
 * the calculation is removed.
*/
public function set_calculation($formula) {
    ...
    $this->calculation_normalized = true;
    return $this->update();
}
```

```
/**
 * Updates this object in the Database, based on its object variables. ID must be set.
*/
public function update($source=null) {
    ...

    // Get the data to update (IDs, column names, values)
    $data = $this->get_record_data();

    // Update the record in the database
    $DB->update_record($this->table, $data);

    ...
}
```

可以看出，“ update ”方法负责使用存储在 object 中的数据来更新数据库。它使用属性“ table ”作为要更新的表名，数据直接来自 object 自己的属性。

所以，基本上，使用 object 注入，我们可以 update 整个数据库中的任何行。我们可以 update 管理员帐户，密码，站点配置，只要是我们想要的。

这里，我们可以 update 一些限制：

update SQL 语句总是以 WHERE 条件结束，并检查我们指定的 ID。 我们不能在我们的数据中利用 SQL 注入。我们设置的值将被转义，并且我们要更新的字段将根据表的实际字段进行检查，因此我们不能使用任何不存在的字段。

只要我们知道要 update 的行 ID，我们就可以 update 我们想要的所有内容。 但是，我们不应该试图更改管理员的密码，或者需要猜测其用户 ID，或者暴力破解数据库中的每个帐户。我们应该尽量减少对服务器的影响。

4、 高效语句构造

我们不想更改系统中每个用户的密码。所以为了绕过，我们可以通过更改存储在“ config ”表中的“ site_admins ”配置值，将我们的用户添加为系统中的另一个管理员。
但是我们如何猜测表中的具体配置的 ID 呢? 好吧，我们不能。但是我们可以尝试改变 WHERE SQL 语句。
为了做到这一点，我们需要使用一个 SQL 注入。由于我们不能利用任何数据字段，我们必须在表名称的本身中插入我们的 SQL 注入，而不是在任何地方转义。
但是，这提出了另一个问题 – 在执行 UPDATE 语句之前，Moodle 正在向数据库查询指定表的列名和类型。
这意味着我们必须在 SELECT 语句和 update 语句上同时执行相同的 SQL 注入，SELECT 语句应该返回表的正确数据， UPDATE 语句应该 update“ site_admins ”配置值。
我们来看看这两个语句：

```
    SELECT column_name, data_type, character_maximum_lengt
h, numeric_precision, numeric_scale, is_nullable, column_
type, column_default, column_key, extraFROM information_s
chema.columnsWHERE table_name = '[TABLE_NAME]'ORDER BY or
dinal_position
```

UPDATE [TABLE_NAME] SET [DATA] WHERE WHERE id='[ID]'
很明显，我们在两个表中唯一的注入点是表名。
利用此 SQL 注入的一种方法是在我们 update 的表之后启动一个多行注释 (/*) ，并将其关闭在我们的一个数据参数中。这样我们的数据可以包含我们改变的 WHERE 语句，通过配置名称而不是它的 ID 过滤数据。 所以，我们的有效载荷是这样的：

```
    SELECT … WHERE table_name = 'config' /*'ORDER BY ordin
al_position

    UPDATE config' /* SET field='*/ SET value=our_user_id
WHERE name=site_admins-- WHERE id='[ID]'
```

但是我们如何在表名中插入一个注释，使 SELECT 语句和 UPDATE 语句都可以正常工作? 显然， UPDATE 语句不会工作，因为在表名后面添加撇号的话，select 语句将不工作，因为我们无法在 SQL 中关闭它时不打开多行注释。
如果我们将多行注释插入到我们的表名中而不关闭字符串，然后再继续使用另一个 OR 语句的 WHERE 条件会怎么样? 我的意思是，这样的：

```
    SELECT … WHERE table_name = 'config /*' or table_name
LIKE '%config'ORDER BY ordinal_position

    UPDATE config /*' or table_name LIKE '%config SET fiel
d='*/ SET value=our_user_id WHERE name=site_admins-- WHER
E id='[ID]'
```

虽然 SELECT 语句有许多 self-explanatory ，但 UPDATE 语句可能需要更多的 explanation。 试试以 MySQL 解析的方式显示它：

```
    UPDATE config /*' or table_name LIKE '%config SET fiel
d='*/ SET value=our_user_id WHERE name=site_admins -- WHE
RE id='[ID]'
```

现在可以清楚地看到，我们已经将表名和我们控制的第一个数据值之间的一切注释掉了。这样，我们能在 UPDATE 语句上使用 SQL 语句，而不影响 SELECT 语句了。
然后我们只需将配置值设置为我们想要的值，最好是我们的用户 ID，然后添加我们改进的 WHERE 语句，根据配置名称过滤表。最后，我们添加一行注释，以删除内置的 WHERE 语句。我们在两个不同的查询上成功利用了相同的 SQL 注入。
所以，我们现在要做的是等待配置缓存刷新，这应该每天都会发生，或者只是强制刷新它自己，通过删除“ allversionshash ”的配置值，它存储系统中所有核心文件的 SHA-1 哈希值。更改此配置的值将使 Moodle 认为它经历了固件更新，并为我们刷新整个缓存。
获得完整的管理员权限后，执行代码就像将新插件或模板上传到服务器一样简单。
所以，在利用一些虚假的假设，一个对象注入，一个双重 SQL 注入和一个允许的管理员仪表板后，我们终于赢了。

5、修复建议

到厂商的注意下载升级补丁以修复这个安全问题：
https://git.moodle.org/gw?p=moodle.git&a=search&h=HEAD&st=commit&s=MDL-58010