Sec Bug #81026 PHP-FPM oob R/W in root process leading to privilege escalation

Submitted: 2021-05-10 09:16 UTC **Modified:** 2021-10-26 11:42 UTC
From: c dot fol at ambionics dot io **Assigned:** bukka (profile)
Status: Closed **Package:** FPM related
PHP Version: 8.0.6 **OS:**
Private report: No **CVE-ID:** 2021-21703

[2021-05-10 09:16 UTC] c dot fol at ambionics dot io

Description:

Hello,

Bug

TL;DR: You can force the root FPM process to read/write at arbitrary locations using pointers located in the SHM, leading to a privilege escalation from www-data to root.

In PHP-FPM, each worker has an associated scoreboard (`fpm_scoreboard_proc_s`) structure, which contains a PID, its current state (is it handling a request or idle ? Which PHP file is it processing ?), and a few stats alongside (memory and CPU time used). Since both the main process and the worker itself need to read and write this structure, it resides in a shared memory mapping.
The main process controls a `fpm_scoreboard_s` structure, also located in the SHM. It contains the number of active workers, and an array of pointers to `fpm_scoreboard_proc_s` structures.

Here's an example of `fpm_scoreboard_s` and `fpm_scoreboard_proc_s`:

```cpp
pwndbg> p fpm_worker_all_pools->scoreboard
$6 = (struct fpm_scoreboard_s *) 0x7ff114945000
pwndbg> xi 0x7ff114945000
Extended information for virtual address 0x7ff114945000:

  Containing mapping:
    0x7ff114945000     0x7ff114947000 rw-p     2000 0      /dev/zero (deleted)

  Offset information:
         Mapped Area 0x7ff114945000 = 0x7ff114945000 + 0x0
         File (Base) 0x7ff114945000 = 0x7ff109a31000 + 0xaf14000
pwndbg> p *fpm_worker_all_pools->scoreboard // fpm_scoreboard_s
$7 = {
  {
    lock = 0,
    dummy = '\000' <repeats 15 times>
  },
  pool = "www", '\000' <repeats 28 times>,
  pm = 2,
  start_epoch = 1619077029,
  idle = 2,
  active = 0,
  active_max = 0,
  requests = 0,
  max_children_reached = 0,
  lq = 0,
  lq_max = 0,
  lq_len = 0,
  nprocs = 5, // Array size
  free_proc = 2,
  slow_rq = 0,
  procs = 0x7ff114945078 // fpm_scoreboard_proc_s*[]
}
pwndbg> p *fpm_worker_all_pools->scoreboard->procs[0] // fpm_scoreboard_proc_s
$8 = {
  {
    lock = 0,
    dummy = '\000' <repeats 15 times>
  },
  used = 1,
  start_epoch = 1619077029,
  pid = 1444,
  requests = 0,
  request_stage = FPM_REQUEST_ACCEPTING,
  ...
  request_uri = '\000' <repeats 127 times>,
  query_string = '\000' <repeats 511 times>,
  request_method = '\000' <repeats 15 times>,
  ...
}
```

As you can see, scoreboard->procs is an array of pointers. Since it is located in the SHM, it can be read/modified by PHP-FPM workers. Since the root process also uses this pointers, this can lead to privilege escalation.

As a quick test, you can set these pointers to `0xAABBCCDDEE` from a worker process and watch PHP-FPM's main process crash while trying to spawn a new one.

Exploit primitives

What does the main process do with these `scoreboard->procs` pointers ? When a worker dies, its scoreboard proc will be marked as unused by the main process using `fpm_scoreboard_proc_free()`. When a new one spawns, `fpm_scoreboard_proc_alloc()` will be used to "link" the new worker to an unused scoreboard.

Let's check `fpm_scoreboard_proc_free()` first: the dying process' PID has been previously obtained by the worker shutdown procedure, and from this a `child_index` has been obtained. It is the index of the dead worker's scoreboard structure in the `scoreboard->procs` array. If this structure's `used` flag is set, it is zeroed: 1168 bytes are set to zero.

When a worker spawns, it needs a `fpm_scoreboard_proc_s` structure. `fpm_scoreboard_proc_alloc()` iterates over

`scoreboard->procs` and find an unused structure (by checking that the `used` field is *zero*), and mark it as used.
This will effectively change a `0` int to a `1`.

These are really bad primitives but this is enough to get a privilege escalation. I can get into more details if you
need to, and/or provide an exploit.

Fix

I might be very wrong, but I have a few ideas since I've read this code a lot.

1. Get rid of pointers

Maybe you could change `scoreboard->procs` to be an array of `fpm_scoreboard_proc_s` instead of a an array of pointers
? Since the array access index is properly checked (for instance in `fpm_scoreboard_proc_free`, child_index is bound-
checked), this would prevent the edition of pointers. No pointers no problems.

2. Get rid of scoreboard->nprocs

You always allocate wp->config->pm_max_children scoreboard procs (that is, scoreboard->nprocs == wp->config-
>pm_max_children). I don't believe workers need to access nprocs. Even if they did, they can't be able to change it.
Otherwise, a worker can set it to a huge value and when the main process tries to go through the ->procs array, it
will read OOB and crash.

Test script:

You can use a PHP sandbox escape exploit to get arbitrary R/W in a worker's memory. Since we don't have this, let's
just use GDB to "emulate" this (you need symbols):

Spawn PHP-FPM and check PIDs

```
$ service php-fpm restart
$ ps faux | grep php-fpm
root        1242  ...  php-fpm: master process (/etc/php/8.0/fpm/php-fpm.conf)
www-data    1320  ...  \_ php-fpm: pool www
www-data    1321  ...  \_ php-fpm: pool www
```

Trigger bug

We'll change the shared memory from the first child (1320):

```
$ sudo gdb -q -p 1320
pwndbg> p fpm_scoreboard->procs[0] = 0x11223344
```

Almost instantly, the root process crashes.

Debug infos

Program received signal SIGSEGV, Segmentation fault.

Backtrace:

```
#0  fpm_request_is_idle (child=child@entry=0x5587c0bf2a10) at ./sapi/fpm/fpm/fpm_request.c:293
#1  0x00005587bfe1e691 in fpm_pctl_perform_idle_server_maintenance (now=0x7ffd515e0af0) at
./sapi/fpm/fpm/fpm_process_ctl.c:328
#2  fpm_pctl_perform_idle_server_maintenance_heartbeat (ev=<optimized out>, which=<optimized out>, arg=<optimized
out>) at ./sapi/fpm/fpm/fpm_process_ctl.c:481
#3  0x00005587bfe1a93a in fpm_event_fire (ev=0x5587bff964a0 <heartbeat>) at ./sapi/fpm/fpm/fpm_events.c:487
#4  fpm_event_loop (err=err@entry=0) at ./sapi/fpm/fpm/fpm_events.c:467
#5  0x00005587bfe14bc7 in fpm_run (max_requests=0x7ffd515e0d7c) at ./sapi/fpm/fpm/fpm.c:113
#6  0x00005587bfbd6a64 in main (argc=argc@entry=4, argv=argv@entry=0x7ffd515e12b8) at ./sapi/fpm/fpm/fpm_main.c:1827
#7  0x00007f4370d6d0b3 in __libc_start_main (main=0x5587bfbd6130 <main>, argc=4, argv=0x7ffd515e12b8, init=<optimized
out>, fini=<optimized out>, rtld_fini=<optimized out>, stack_end=0x7ffd515e12a8) at ../csu/libc-start.c:308
#8  0x00005587bfbd7e5e in _start () at ./sapi/fpm/fpm/fpm_main.c:1151
```

Crash RIP and RAX:

```
0x5587bfe1f8d0 <fpm_request_is_idle+32>       cmp    dword ptr [rax + 0x30], 1

RAX == 0x11223344
```

Patches

fpm_scoreboard_proc_oob_fix_v4.patch (last revision 2021-10-10 19:20 UTC by bukka@php.net)
fmp_scoreboard_proc_oob_v3.patch (last revision 2021-05-31 21:55 UTC by bukka@php.net)
~~fmp_scoreboard_proc_oob_v2.patch~~ (last revision 2021-05-31 21:16 UTC by bukka@php.net)
fmp_scoreboard_proc_oob_v1.patch (last revision 2021-05-31 13:32 UTC by bukka@php.net)

Add a Patch

Pull Requests

Pull requests:

- Fix bug #81026 (PHP-FPM oob R/W in root process leading to priv escal… (php-src/7614)

Add a Pull Request

History

All | Comments | Changes | Git/SVN commits | Related reports

[2021-05-18 09:14 UTC] c dot fol at ambionics dot io

Hello,

A small update.
The bug has been present since the first fpm_scoreboard implementation (PHP 5.3.7). I now have a reliable exploit for
PHP 8+, which I can provide if needed.

[2021-05-26 09:53 UTC] c dot fol at ambionics dot io

Hello, any update on this ?

[2021-05-26 19:41 UTC] stas@php.net

-Assigned To:
+Assigned To: bukka

[2021-05-31 13:32 UTC] bukka@php.net

The following patch has been added/updated:

Patch Name: fmp_scoreboard_proc_oob_v1.patch
Revision: 1622467931
URL: https://bugs.php.net/patch-display.php?bug=81026&patch=fmp_scoreboard_proc_oob_v1.patch&revision=1622467931

[2021-05-31 13:34 UTC] bukka@php.net

Hi, yeah this look like an issue. I have attached a dirty patch (needs more work) if you could check if it fixes the problem. Please let me know and if good I will polish it...

[2021-05-31 13:37 UTC] c dot fol at ambionics dot io

Hello,

I can't view the patch: "You have no access to bug #81026".
Will you require a CVE for this, or should I ?

Have a great day!
Charles Fol

[2021-05-31 13:37 UTC] c dot fol at ambionics dot io

Related To: Bug #81026

[2021-05-31 13:55 UTC] bukka@php.net

There is a mistake in the v1 patch. The proc_end part is wrong - should replace scoreboard size with proc_start. Will send new patch later.

[2021-05-31 16:15 UTC] cmb@php.net

@bukka, the bug reporter can't see attached patches. It is usually preferable to post patches as secret Gists[1], and to post the link here. If the patch is short, just posting it inline is also fine.

@c dot fol: CVEs are assigned by the PHP security team.

[1] <https://gist.github.com/>

[2021-05-31 21:16 UTC] bukka@php.net

The following patch has been added/updated:

Patch Name: fmp_scoreboard_proc_oob_v2.patch
Revision: 1622495810
URL: https://bugs.php.net/patch-display.php?bug=81026&patch=fmp_scoreboard_proc_oob_v2.patch&revision=1622495810

[2021-05-31 21:19 UTC] bukka@php.net

Fixed patch gist: https://gist.github.com/bukka/72a0408e027b85e72d6f4e622974eb01

[2021-05-31 21:55 UTC] bukka@php.net

The following patch has been added/updated:

Patch Name: fmp_scoreboard_proc_oob_v3.patch
Revision: 1622498128
URL: https://bugs.php.net/patch-display.php?bug=81026&patch=fmp_scoreboard_proc_oob_v3.patch&revision=1622498128

[2021-06-01 08:56 UTC] c dot fol at ambionics dot io

Hello,

-- 1 -------------

The patch does not fix every read/write access to scoreboard->procs[].

Example funcs:

fpm_scoreboard_proc_alloc() -> scoreboard->procs[i]->used = 1;
fpm_scoreboard_proc_free() -> memset(scoreboard->procs[child_index], 0, sizeof(struct fpm_scoreboard_proc_s));

-- 2 -------------

Also, there is a mistake in the patch: fpm_scoreboard_proc_get_from_child() tries to find the lower and upper bounds for ptr addresses, but the computation of the lower bound is wrong:

ptrdiff_t proc_end = proc_start + (wp->config->pm_max_children - 1) * sizeof(struct fpm_scoreboard_proc_s *);

The last operand should be sizeof(struct fpm_scoreboard_proc_s), not sizeof(struct fpm_scoreboard_proc_s *)

-- 3 ------------

Lastly, I kindly reiterate my suggestion of making scoreboard_s.procs an array of fpm_scoreboard_proc_s instead of an array of *pointers* to fpm_scoreboard_proc_s. In the current design, every time you want to access a scoreboard_proc, you will use its index (child->scoreboard_i). As a consequence, the change won't cause a loss of usability.

Even better, you don't have to change the prototype of fpm_scoreboard_proc_get() !

You'd have (pseudo-ish code):

```
struct fpm_scoreboard_s {
        ...
        struct fpm_scoreboard_proc_s procs[];
};

int fpm_scoreboard_init_main() /* {{{ */
{
        ...
        scoreboard_size        = sizeof(struct fpm_scoreboard_s) + (wp->config->pm_max_children) * sizeof(struct
fpm_scoreboard_proc_s);
        ...
}

struct fpm_scoreboard_proc_s *fpm_scoreboard_proc_get(struct fpm_scoreboard_s *scoreboard, int child_index) /* {{{*/
{
        ...
        return &scoreboard->procs[child_index]; // no prototype change, we just return pointer
}
/* }}} */
```

Charles

[2021-06-01 10:14 UTC] bukka@php.net

Yeah I wanted to first try approach that wouldn't change the current paths for normal run that much as it's meant for the branch with security only fixes (PHP-7.3). But missed that alloc and free parts which would make it much messier looking at it. Will have to change the structure of the procs as you say though. Will also need to put together some tests for this which will be the hardest part so will take some time.

There's not that much rush though as we classify those as a low security impact because one needs to have access to the worker first. Basically it's a problem just for the shared hostings but most users shoudl not be impacted. Still security issue though.

[2021-06-01 11:52 UTC] c dot fol at ambionics dot io

"There's not that much rush though as we classify those as a low security impact because one needs to have access to the worker first. Basically it's a problem just for the shared hostings but most users shoudl not be impacted."

I disagree on both points.

I understand PHP's stance on "sandbox escape" exploits: if you can run PHP code, you should be able to run machine code, and mitigation such as disable_functions and safe mode and here to protect against remote exploits. In other words, the logic is that PHP does not act as a sandbox when you can run arbitrary (PHP) code. However, in this case, the presence of the PHP-FPM application poses a new, *otherwise-nonexistent*, security risk to the server.

Furthermore, local root exploits are never referred to as "low security impact". CVE-2019-0211, targeting software relative to the same use (Apache HTTPd), scores a 7.8 CVSS (high).
Saying it only affects shared hosting is a huge understatement.

My 2 cents,
Charles

[2021-06-01 16:59 UTC] bukka@php.net

Yeah it's probably more medium so it should get CVE. Think high is reserved just for exploit without any access but not 100% sure. It doesn't really matter that much anyway as it doesn't get it fixed faster. I will try to find some time to put together a patch and a test but it will probably take few weeks as my time is pretty limitted.

[2021-06-02 06:01 UTC] stas@php.net

```
-CVE-ID:
+CVE-ID: 2021-21703
```

[2021-06-05 20:20 UTC] bukka@php.net

So I had a bit closer look what needs to be changed and it's actually worse than I thought. The thing is that scoreboard->nprocs cannot be trusted as well so we will have to pass a worker pool instead of scoreboard where it's used by the master. The whole thing is touching almost every part of the scoreboard usage (process idle timeout, pool creation / clean up / scaling, full status and possibly some other things). Unfortunately we don't have any test coverage for most of those parts. So we need to improve the test coverage first before getting this fixed to minimise risk that the actual fix breaks anything. It might take a bit of time though.

[2021-06-06 07:42 UTC] c dot fol at ambionics dot io

Yes, as I stated in my first report, you have to get rid of nprocs or keep it out of the SHM. Also, two other problems I saw (although less impactful): a worker can cause a DOS by setting all scoreboard_proc_s.used to 1 (or even -1 to avoid races) or scoreboard(_proc)_s.lock to 1.

I agree that's there's going to be work, sadly. Good luck !

[2021-06-30 12:16 UTC] c dot fol at ambionics dot io

Hello PHP team,

Any update on this ? Can we be of any help ?

Best regards,

[2021-07-25 20:48 UTC] bukka@php.net

I have been quite busy but just pushed some progress on the process idle test that I want to do first. Once it's done, I will take a look into the scoreboard refactoring. Might take me some time though.

[2021-09-05 18:38 UTC] bukka@php.net

Just an update about progress on this. I have just created a PR for process idle timeout that should improve the test coverage of the parts that will need to be changed: https://github.com/php/php-src/pull/7464 . I will start looking on those changes as the next thing when I got time for FPM.

Also I decided to target only 7.4+ as 7.3 will be soon out of security support and the changes will be relatively invasive and don't want to cause any regressions that won't be possible to fix once it gets out of support.

[2021-09-13 07:40 UTC] c dot fol at ambionics dot io

Hello bukka,

Thanks for the update. Here's an update on our side: we'll be revealing the bug on a french infosec conference on October 15th, 2021. A blogpost will come out afterwards, and the exploit (a bit later) as well.

[2021-09-19 19:54 UTC] bukka@php.net

Not sure if it's going to be fixed by then. It would be nice to get a heads up sooner as I have got just weekends and prioritise some other work as well. If I knew that date sooner, it could be potentially fixed sooner. Also our releases take longer so there might be even less time to get it out.

I will be hopefully looking into it next week so will see what I can do.

[2021-10-10 19:20 UTC] bukka@php.net

The following patch has been added/updated:

Patch Name: fpm_scoreboard_proc_oob_fix_v4.patch
Revision: 1633893616
URL: https://bugs.php.net/patch-display.php?bug=81026&patch=fpm_scoreboard_proc_oob_fix_v4.patch&revision=1633893616

[2021-10-10 19:26 UTC] bukka@php.net

Hello, just attached a new patch fpm_scoreboard_proc_oob_fix_v4.patch . It changes the scoreboard structure to eleminate access through the pointer and changes the way how the scoreboard is retrieved in the master process (so it doesn't use the nprocs from the scoreboard but instead uses the configured value for the max children). I run all the tests and it seems to work but not 100% sure if I covered all edge cases. So review and testing would be highly appreciated.

[2021-10-10 20:42 UTC] c dot fol at ambionics dot io

Hello,

First of, sorry for the late warning, I told you as soon as I knew.
I still can't access the patch file ("You have no access to bug #81026").

Charles

[2021-10-10 20:42 UTC] c dot fol at ambionics dot io

Related To: Bug #81026

[2021-10-11 04:53 UTC] stas@php.net

Submitter view does not allow access to patches? That's weird, must be a bug. Anyway, in this case I recommend using secret gists, that should be secure enough for this case. See:
https://gist.github.com/smalyshev/ee7779f098cbfe2fab3799ada2a2f0f7

[2021-10-11 04:55 UTC] stas@php.net

If the patch works, we could have it in the next PHP release which is planned for Oct 21.

[2021-10-11 10:48 UTC] c dot fol at ambionics dot io

Hello,

Thanks for the GIST. The patch gets heavily rejected when I apply it with "git apply", when working on commit "35adc4f7f613a036264272bbd9e0629b9b91bfb8", which is the last commit of the master branch.
Can you give me either the commit ID I need to apply the patch on, or the full PHP core source with patch applied ?

Charles

[2021-10-11 11:09 UTC] bukka@php.net

Please try to apply against PHP-7.4 which is the targeted version for this patch.

[2021-10-11 11:59 UTC] c dot fol at ambionics dot io

OK I managed to have the patch work OK-ish.

As far as I understand, the only potentially unsafe usage of "->nprocs" is in fpm_scoreboard_proc_get(), but it should not matter because the main process won't use fpm_scoreboard_proc_get().

Regarding the mutation of procs from an array of pointers to a "simple" array, I think this is the correct solution, and it should work well.

So in my point of view, the patch is good.

On another note, I'd like to put your attention on Sec Bug #81513, which is the second bug I used to exploit the privilege escalation. This one is easy to patch, and kinda complement this one in regards to the final impact: escalating privileges from www-data to root.

In any case, I apologize again for the short delay. We won't release the blogpost until after this gets patched (hopefully on October 21st)

Best regards,
Charles

[2021-10-11 20:39 UTC] bukka@php.net

Thanks for the review. I just emailed Nikita if he could also take a look so hopefully he will find some time. I will also try to do a bit more manual testing.

@Stas What's the latest to get it ready for October 21st release? I mean when does it need to get merged to get it to that release? Thanks

[2021-10-12 01:34 UTC] stas@php.net

I usually merge stuff on Sunday/Monday prior, so that'd be Oct 18th.

[2021-10-12 14:24 UTC] nikic@php.net

I'm not super familiar with this area, but the patch looks reasonable to me. It's a bit unfortunate that nprocs still remains, but I guess avoiding it would require threading things through more functions.

Just out of interest, I assume the general issue here does not affect opcache SHM because it is not initialized in the root process? Opcache SHM is not (and will definitely never be) safe against malicious modification.

[2021-10-12 20:59 UTC] bukka@php.net

Technically there's not much need to not use nprocs if it's accessed only in the worker context. I think that's kind of the same as opcache shm which should be also only used by workers. It means that we can't protect crashing / impacting of other precesses even in different pool most likely. But what we can do is to at least protect the master process which means prevent possibility of privilege escalation as it often runs as a root.

[2021-10-17 19:25 UTC] bukka@php.net

Hi Stas, so I have done a bit more testing and haven't found any issue so think it's probably good to go.

I'm not exactly sure about handling of security fixex as I think it's usually handled by RMs / you. This one is just for 7.4 and up as it's too risky for 7.3 so I could technically merge it myself. If you want me to do it, please let me know and I will do it.

Otherwise if you want to apply it yourself, please apply the patch to PHP-7.4 like

```
git am --signoff < fpm_scoreboard_proc_oob_fix_v4.patch
```

It should work fine with PHP-7.4 (I tested that). I also checked the merges and there will be one conflict in fpm_scoreboard.h when you merge to PHP-8.0 which should look like this

```
diff --cc sapi/fpm/fpm/fpm_scoreboard.h
index 060eddea98,9d5981e1c7..0000000000
--- a/sapi/fpm/fpm/fpm_scoreboard.h
+++ b/sapi/fpm/fpm/fpm_scoreboard.h
@@@ -63,8 -63,7 +63,12 @@@ struct fpm_scoreboard_s
        unsigned int nprocs;
        int free_proc;
        unsigned long int slow_rq;
++<<<<<<< HEAD
 +      struct fpm_scoreboard_s *shared;
 +      struct fpm_scoreboard_proc_s *procs[];
++=======
+       struct fpm_scoreboard_proc_s procs[];
++>>>>>>> PHP-7.4
  };

  int fpm_scoreboard_init_main();
```

That needs a manual resolution to look like

```
struct fpm_scoreboard_s *shared;
struct fpm_scoreboard_proc_s procs[];
```

It means keeping the shared pointer but changing the procs. So after fixing the git diff should look like this:

```
diff --cc sapi/fpm/fpm/fpm_scoreboard.h
index 060eddea98,9d5981e1c7..0000000000
--- a/sapi/fpm/fpm/fpm_scoreboard.h
+++ b/sapi/fpm/fpm/fpm_scoreboard.h
@@@ -63,8 -63,7 +63,8 @@@ struct fpm_scoreboard_s
        unsigned int nprocs;
        int free_proc;
        unsigned long int slow_rq;
 +      struct fpm_scoreboard_s *shared;
-       struct fpm_scoreboard_proc_s *procs[];
+       struct fpm_scoreboard_proc_s procs[];
  };
```

Once committed, further merges to PHP-8.1 and master should be fine.

[2021-10-18 03:13 UTC] stas@php.net

I can merge 7.4 and up, but would be nice if we also could do 7.3 too since it's still supported for security fixes.

[2021-10-18 03:29 UTC] stas@php.net

The patch seems to apply cleanly to 7.3 too, so I think we can keep it there, unless I hear any objections. I don't see much changes that happened in the affected area between 7.3 and 7.4, even though I obviously could miss something as I don't know that code very well. I have the patches ready and will push them on Monday morning (Pacific time).

[2021-10-18 03:31 UTC] stas@php.net

On the second thought, I'll push 7,4+ and will wait for a word from Jakub on 7.3.

[2021-10-18 09:14 UTC] bukka@php.net

I think it's too risky for PHP 7.3 if there's any issue. It goes out of support soon so I don't think it's worth it. It's relatively big patch touching scoreboard which is the main shared piece in FPM so if something goes wrong, then it's quite a big problem.

Considering the attack surface that requires someone to have access to the environment or at most to the code (if exec and like allowed), then the number of users where this might be really problem should be quite small - personally I have never seen such setuo but possibly some shared hostings could be impacted. The thing is that they most likely use distro package or might just apply the patch themself on their own risk. Also distros can apply the patch on their supported versions too if they wish - it's less problem for them as they have got longer support and might potential apply further fixes in case something goes wrong here.

[2021-10-18 09:37 UTC] bukka@php.net

Btw I see that Nikita caused us another conflict by merging this https://github.com/php/php-src/commit/f71810fb6f98251ace18ba49bc269d19ab27579a ... :)

You can fix manauly by applying changes from both so the params in both calls look like

wp->config->name, (int) pid, buf, (long) tv2.tv_sec, (int) tv2.tv_usec

[2021-10-18 09:41 UTC] bukka@php.net

That conflict is for merging to PHP-8.1...

[2021-10-18 12:33 UTC] c dot fol at ambionics dot io

```
"""
Considering the attack surface that requires someone to have access to the environment or at most to the code (if exec and like allowed), then the number of users where this might be really problem should be quite small - personally I have never seen such setuo but possibly some shared hostings could be impacted.
"""
```

Once again, I believe you're underestimating the impact of this bug. Most distros make the PHP-FPM service as root straight up when you install it through package managers. This means that in most cases, remote code execution vulnerabilities on PHP-FPM (NGINX, IIS) will now yield a root shell instead of the standard www-data. I've been a pentester for a long time, and I can tell you this greatly improves the attack surface for an attacker.

I know I'm insisting a lot -- this is the last time I interject, but this bug is scary, and damage will be done if people cannot patch their PHP-FPM easily.

Therefore, I really think this needs a proper fix for all supported PHP versions.

Best regards,
Charles

[2021-10-18 14:28 UTC] bukka@php.net

Well yeah in combination with another execution vulnerability it will be more problematic and it's true that there are still legacy setups with world exposed web server and other services running in the same box. However most of such old setups are often behind LB and running PHP with Web server in private network and probably even more setups nowadays are running PHP in container and communicating over the network with web server in another container. What I'm saying is that it's usually quite hard to get to the PHP server that would run PHP 7.3 (there are probably still plenty if such services running older versions but those that did the update and using something like this will be minority IMO).

As I said even if there are such PHP 7.3 users, those will most likely use distro packages. Distro can easily apply the patch and possibly follow up. But what I don't want to do is patch version that has got maybe one or max two other releases before going to complete freeze with such a big patch to the critical part of FPM and after some time find out that there some issues that can impact more users that actually run 7.3 and this issue is not problem for them. Hope that makes sense.

[2021-10-18 22:40 UTC] git@php.net

Automatic comment on behalf of bukka (author) and smalyshev (committer)
Revision: https://github.com/php/php-src/commit/cb2021e5f69da5e2868130a05bb53db0f9f89e4b
Log: Fix bug #81026 (PHP-FPM oob R/W in root process leading to priv escalation)

[2021-10-18 22:40 UTC] git@php.net

```
-Status: Assigned
+Status: Closed
```

[2021-10-18 22:40 UTC] git@php.net

Automatic comment on behalf of bukka (author) and smalyshev (committer)
Revision: https://github.com/php/php-src/commit/fadb1f8c1d08ae62b4f0a16917040fde57a3b93b
Log: Fix bug #81026 (PHP-FPM oob R/W in root process leading to priv escalation)

[2021-10-24 16:26 UTC] chris at cretaforce dot gr

Will the patch land to 7.3 ?

[2021-10-25 13:38 UTC] phpnet at nuvini dot com

Hello,

I'm not too much into PHP internals (just reading the changelog with this release) and to me from what I read this seems like a massive security impact for us. We have hundreds of servers (DirectAdmin servers and other shared hosting servers) where the PHP-FPM master runs as root and we create worker pools with reduced privileges.

As far as I can see once the exploit for this is released this is an easy way to get root on all those servers by any user that can upload PHP files and run them.

For PHP 7.3 (and older) that won't be patched, is there any workaround available? A setting in the config that when set prevents the required conditions for exploiting this?

[2021-10-25 19:25 UTC] stas@php.net

The patch for 7.4 applies cleanly to 7.3, so you could just use that patch. Specifically, it's this one:
https://github.com/php/php-src/commit/cb2021e5f69da5e2868130a05bb53db0f9f89e4b

[2021-10-25 19:30 UTC] stas@php.net

As for older versions, since they're out of support, your only options would be either to try and port the patch by yourself, or have somebody - like a distro if any support older versions, or a commercial company offering long-term old versions support - port it for you. It's not feasible for the PHP project to support EOLed versions, we just don't have the resources for it.

[2021-10-25 20:42 UTC] bukka@php.net

It might even apply to the older versions as well as there haven't been too many changes so if you are running your compiled version of PHP, it shouldn't be that difficult.

If you are on a distro package, then I think that every sane distro should apply it to the their supported lower versions so best to check with your distro packager.

[2021-10-25 20:47 UTC] bukka@php.net

Also the impact might be reduced but restricting shell access and prohibiting all Program execution Functions (exec, system and so on). Although not 100% if that's enough. Maybe Charles can shed a bit more light if that could help.

[2021-10-26 08:32 UTC] derick@php.net

IMO the argument that it is quite invasive is valid...

However, I think we should merge it into PHP 7.3, and also make the guarantee to continue providing support in the form of bug and security fixes to the *PHP-FPM* part for a little extended end-of-live (beyond Dec 6th).

[2021-10-26 08:41 UTC] krakjoe@php.net

In the interest of building a consensus ...

I'm in favor of merging into 7.3, and endorse the strategy that it is better to be prepared to make further releases in the unlikely event that a problem is found.

I hear the arguments against doing this but remain unconvinced that they justify effectively shortening security support for this branch without notice; To me it makes more sense to be ready to extend that support period, if we must.

[2021-10-26 08:43 UTC] remi@php.net

I also think this fix should be applied in 7.3 branch

FYI this is already applied on various distribution packages repo (Remi's RPM, Ondrej's DEB, ...) and after thousands of downloads, nobody have report any issue.

[2021-10-26 09:04 UTC] cmb@php.net

The following pull request has been associated:

Patch Name: Fix bug #81026 (PHP-FPM oob R/W in root process leading to priv escal…
On GitHub: https://github.com/php/php-src/pull/7614
Patch: https://github.com/php/php-src/pull/7614.patch

[2021-10-26 09:56 UTC] c dot fol at ambionics dot io

Hello,

"""
Also the impact might be reduced but restricting shell access and prohibiting all Program execution Functions (exec, system and so on). Although not 100% if that's enough. Maybe Charles can shed a bit more light if that could help.
"""

It does not. To exploit you need to escape PHP's sandbox anyway, so limitations such as disable_functions or open_basedir are no use, sadly.

Hope this helps,
Charles

[2021-10-26 10:08 UTC] bukka@php.net

Ok it makes sense to back port to 7.3 in that case.

[2021-10-26 10:16 UTC] bukka@php.net

Hi Charles

> It does not. To exploit you need to escape PHP's sandbox anyway, so limitations such as disable_functions or open_basedir are no use, sadly.

It's just for my interest but can you explain how can you escape PHP's sandbox without any of the program execution function and without any shell access (e.g. ssh) to the machine where the PHP is running?

[2021-10-26 10:54 UTC] c dot fol at ambionics dot io

Hello,

First, to trigger this bug, you need to be able to execute PHP code. For instance, as an attacker, you can reach something such as eval($_POST['stuff']). Evidently, this is more convoluted in practice: maybe you can overwrite part of a PHP file, maybe you can reach unserialize($controlled_data), etc. So, this is the starting point: you control some PHP code on the server.

Then, you need to use a binary bug in the PHP interpreter to "escape". This could be a use after free, a buffer overflow, a type confusion...

Imagine you have an internal PHP function (thus in C) which takes a zval as argument. The sole goal of this (imaginary) function is to add an element into the zval, which is supposed to be an array.

The code would look like this:

```
PHP_FUNCTION(add_some_value_into_array)
{
    ZEND_PARSE_PARAMETERS_START(1, 1)
                Z_PARAM_ZVAL(my_zval)
        ZEND_PARSE_PARAMETERS_END();

    // Add element at index 0 in the array
    add_index_string(HASH_OF(my_zval), 0, "hello !");

    RETURN_TRUE;
}
```

Unluckily, the programmer forgot to add the check if(Z_TYPE_P(my_zval) == IS_ARRAY) before using the zval as an array.

Therefore, a local attacker can call: add_some_value_into_array(0x1234).

When PHP tries to use my_zval (which contains a zend_long) as an array, it will dereference my_zval.val.arr, which is in fact my_zval.val.long.

So the C pointer it tries to dereference is 0x1234...

Therefore, as an attacker, I can make PHP use an arbitrary pointer as an array. This gives me total control over the zend_array contents, and most notably its zend_array.pDestructor element, which is a function pointer.

Now, let's say I built a fake array which contains a single element, at index 0. When add_index_string gets called, $arr[0] will get deleted before "hello !" is added. Therefore, zval.value.arr->pDestructor(zval.value.arr->arData[0]) (or something like that) gets called. Since I control both these values, I can now make PHP call an arbitrary C function with an arbitrary argument. That's the "escape" of PHP's sandbox. You're not executing arbitrary PHP anymore, you're executing machine instructions. Protections such as disable_functions and open_basedir have no effect here.

This gives the attacker complete control over the program, even though s/he hasn't used any "code execution" PHP function such as system, exec, etc. It's even better: s/he can now access the whole memory of the current process, which s/he wouldn't be able to do if I just executed another program with PHP's system().

This is the starting point of the exploit for PHP-FPM, where you need to access a worker's SHM.

The sandbox escape bug I used in reality is https://github.com/cfreal/exploits/blob/master/php-SplDoublyLinkedList-offsetUnset/exploit.php

You can read the header comments to understand the bug.

Hope this makes sense,
Charles

[2021-10-26 11:42 UTC] bukka@php.net

Ah ok that makes sense now. I didn't realised that are already such triggers like SplDoublyLinkedList. Damn C, we should probably rewrite everything in Rust... :)

[2021-10-26 14:12 UTC] git@php.net

Automatic comment on behalf of bukka (author) and cmb69 (committer)
Revision: https://github.com/php/php-src/commit/f47798e685e5c97ba52fe7c9bf408a5b92ab5b6a
Log: Fix bug #81026 (PHP-FPM oob R/W in root process leading to priv escalation)

[2021-10-27 13:54 UTC] chris at cretaforce dot gr

This patch works for 5.6:

https://github.com/remicollet/php-src-security/commit/0f2a7ebf53b3787651157f1509c0cd6fc1292a1a.patch