buffer overflow #1799

Closed rain6851 opened this issue on Apr 15, 2020 · 9 comments

rain6851 commented on Apr 15, 2020

Enviroment

```
operating system: ubuntu18.04
compile command:  build with [ASAN](https://github.com/google/sanitizers)

test command: ./espruino poc
```

poc:

```
// Socket server and client test

var result = 0;
var port = 41234;
let dgram = require('dgram');

let srv = dgram.createSocket('udp4');
srv = srv.bind(port, function() {
  srv.on('message', function(msg, info) {
    console.log("<"+JSON.stringify(msg));
    console.log("<"+JSON.stringify(info));
    srv.send(msg='!', info.port, info.address);
  });
});
srv.on('close', function() {
  console.log('server disconnected');
});

let client = dgram.createSocket('udp4');
client.on('message', function(msg, info) {
  console.log(">"+JSON.stringify(msg));
  console.log(">"+JSON.stringify(info));

  result = msg=="42!" && info.address=="127.0.0.1" && info.port==port;

  clearTimeout(failTimeout); // stop the fail fast

  srv.close();
  client.close();
});
client.on('close', function() {
  console.log('client disconnected');
});

// fail the test fast if broken
failTimeout = setTimeout(function() {
  client.close();
  srv.close();
}, 100);
client.send('42', port, 'localhost');
```

vulnerability description:

The poc will cause the memory corruption of the parser. Below is the output of ASAN :

```
---------------------------------
----------------------------- TEST es

 ____                _
|  __|___ ___ ___ _ _|_|___ ___
|  __|_ -| . |  _| | | |   | . |
|____|___|  _|_| |___|_|_|_|___|
         |_| espruino.com
 2v04 (c) 2019 G.Williams

Espruino is Open Source. Our work is supported
only by sales of official boards and donations:
http://espruino.com/Donate

<"42"
<{"address":"127.0.0.1","port":45749,"size":2}
>=================================================================
==125126==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7ffe7b7767c9 at pc 0x000000441623 bp 0x7ffe7b776650 sp 0x7ffe7b776640
                                                         WRITE of size 1 at 0x7ffe7b7767c9 thread T0
#0 0x441622 in jsvGetStringChars src/jsvar.c:1295
#1 0x541020 in socketReceivedUDP libs/network/socketserver.c:392
#2 0x54150a in socketReceived libs/network/socketserver.c:417
#3 0x5431dd in socketClientConnectionsIdle libs/network/socketserver.c:710
#4 0x543e4f in socketIdle libs/network/socketserver.c:837
#5 0x6b66c1 in jswrap_net_idle libs/network/jswrap_net.c:31
#6 0x50fae6 in jswIdle gen/jswrapper.c:1844
#7 0x4e279d in jsiIdle src/jsinteractive.c:2117
#8 0x4e391d in jsiLoop src/jsinteractive.c:2204
```

```
  #9 0x64ad56 in run_test targets/linux/main.c:74
  #10 0x406fe9 in main targets/linux/main.c:287
  #11 0x7f0473ff482f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x2082f)
  #12 0x408178 in _start (/home/node/Espruino/espruino+0x408178)

   Address 0x7ffe7b7767c9 is located in stack of thread T0 at offset 105 in frame
   #0 0x540d8f in socketReceivedUDP libs/network/socketserver.c:387

   This frame has 2 object(s):
    [32, 48) 'args'
    [96, 105) 'buf' <== Memory access at offset 105 overflows this variable
    HINT: this may be a false positive if your program uses some custom stack unwind mechanism or swapcontext
                          (longjmp and C++ exceptions *are* supported)
    SUMMARY: AddressSanitizer: stack-buffer-overflow src/jsvar.c:1295 jsvGetStringChars
    Shadow bytes around the buggy address:
       0x10004f6e6ca0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
       0x10004f6e6cb0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
       0x10004f6e6cc0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
       0x10004f6e6cd0: f1 f1 f1 f1 00 00 00 00 00 f4 f4 f4 f3 f3 f3 f3
       0x10004f6e6ce0: 00 00 00 00 00 00 00 00 00 00 00 00 f1 f1 f1 f1
  =>0x10004f6e6cf0: 00 00 f4 f4 f2 f2 f2 f2 00[01]f4 f4 f3 f3 f3 f3
       0x10004f6e6d00: 00 00 00 00 00 00 00 00 00 00 00 00 f1 f1 f1 f1
       0x10004f6e6d10: 00 00 f4 f4 f3 f3 f3 f3 00 00 00 00 00 00 00 00
       0x10004f6e6d20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
       0x10004f6e6d30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
       0x10004f6e6d40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
      Shadow byte legend (one shadow byte represents 8 application bytes):
      Addressable:           00
      Partially addressable: 01 02 03 04 05 06 07
      Heap left redzone:       fa
      Heap right redzone:      fb
      Freed heap region:       fd
      Stack left redzone:      f1
      Stack mid redzone:       f2
      Stack right redzone:     f3
      Stack partial redzone:   f4
      Stack after return:      f5
      Stack use after scope:   f8
      Global redzone:          f9
      Global init order:       f6
      Poisoned by user:        f7
     Container overflow:      fc
     Array cookie:            ac
     Intra object redzone:    bb
     ASan internal:           fe
  ==125126==ABORTING
```

gfwilliams commented on Apr 16, 2020 Member

Thanks! However are you sure about this? Can you provide me more detailed info about exactly what line fails? And on an up to date build from Git master?

Looking at the code:

```
char buf[sizeof(JsNetUDPPacketHeader)+1]; // trailing 0 from jsvGetStringChars
jsvGetStringChars(*receiveData, 0, buf, sizeof(JsNetUDPPacketHeader)+1);
```

`jsvGetStringChars` gets *just* the length of data that'll fit in `buf`. Just tested in a debugger and it's fine so I don't see where the out of bounds write is.

rain6851 commented on Apr 23, 2020 • edited Author

@gfwilliams

detection instructions

In some cases, the overflow will not cause the program to crash. However, it can be found through the tool Address sanitizer(https://github.com/google/sanitizers/wiki/AddressSanitizer)

you should add some flags to CFLAGS and DFLAGS in makefile:
The following are the changes I made to the Makefile regarding the compilation options

CFLAGS?=-Wall -Wextra -Wconversion -Werror=implicit-function-declaration -fno-strict-aliasing -g -fsanitize=address
LDFLAGS?=-Winline -g -fsanitize=address

compile command

make

test command

./espruino poc

gfwilliams commented on Apr 23, 2020 Member

Sorry - if you can reproduce maybe you can fix it or at least point me to the actual line of code and what the variables are all set to.

I don't see how this error can occur, I've tested with a normal compile and everything looks spot on.

rain6851 commented on Apr 23, 2020 Author

> Sorry - if you can reproduce maybe you can fix it or at least point me to the actual line of code and what the variables are all set to.
>
> I don't see how this error can occur, I've tested with a normal compile and everything looks spot on.

Vulnerabilities are generally difficult to discover, otherwise they will not be hidden for so long. If I have the time to help you with specific analysis, under normal compilation options, it really does not go wrong.

gfwilliams closed this as completed on Apr 23, 2020

---

rain6851 commented on Apr 23, 2020 • edited Author

@gfwilliams I have helped you clarify your code and the problems in your code. What's more, I propose a fix. This is a typical off-one-byte overflow. If you do not pay attention, it will lead to remote code execution: https://csl.com.co/en/off-by-one-explained/. This kind of overflow is not easy to cause the program to crash directly. It can be well detected by ASAN. I hope you will pay attention to it.

stack overflow process

the smallest poc code

```
let dgram = require('dgram');
let srv = dgram.createSocket('udp4');
srv = srv.bind(port, function() {
  srv.on('message', function(msg, info) {
     srv.send(msg=" ", info.port, info.address); // this line cause overflow
  });
});
let client = dgram.createSocket('udp4');
client.send('42', port, 'localhost');
```

reason

When the code execute to src/jsvar.c: 1286
The context is as follows:

```
$13 = {
  charIdx = 0x0,
  charsInVar = 0xa,
  varIndex = 0x0,
  var = 0x7ffff7f6b0a0,
  ptr = 0x7ffff7f6b0a0 "\177"
}
gef?? print len
$14 = 0x9
gef?? print str
$15 = 0x7fffffffdda0 "\n"
```

The funcion `jsvStringIteratorHasChar` will iterate until `charsInVar` = 0x9. For the reason of the code below:(src/jsvariterator.h:64)

```
static ALWAYS_INLINE bool jsvStringIteratorHasChar(JsvStringIterator *it) {
  return it->charIdx < it->charsInVar;
}
```

When the code execute to 1295(src/jsvar.c), the contents of `it` and `len` are shown in below:

```
gef?? print it
$2 = {
  charIdx = 0x0,
  charsInVar = 0x9,
  varIndex = 0x0,
  var = 0x7ffff7f6b4c0,
  ptr = 0x7ffff7f6b4c0 "\177"
}
gef?? print len
$3 = 0x9
```

In the current situation, `str` has exceeded the maximum length of buf by one byte. Because of that `charsInVar` = 0x9 and `len` = 0x9.
When the engine execute the code `*str =0`, it will occur overflow one byte.

How the fix

Apply for one more byte for buf.
The code in libs/network/socketserver.c:387 should be:

```
void socketReceivedUDP(JsVar *connection, JsVar **receiveData) {
  // Get the header
  size_t len = jsvGetStringLength(*receiveData);
  if (len < sizeof(JsNetUDPPacketHeader)) return; // not enough data for header!
  char buf[sizeof(JsNetUDPPacketHeader)+2]; // trailing 0 from jsvGetStringChars
  jsvGetStringChars(*receiveData, 0, buf, sizeof(JsNetUDPPacketHeader)+1);
```

gfwilliams commented on Apr 24, 2020 Member

Perfect, thanks for this! Looks like a fix is needed in `jsvGetStringChars` for the case where the data length is exactly that size - this will affect not just UDP but a whole bunch of stuff

gfwilliams reopened this on Apr 24, 2020

rain6851 commented on Apr 24, 2020 Author

> Perfect, thanks for this! Looks like a fix is needed in `jsvGetStringChars` for the case where the data length is exactly that size - this will affect not just UDP but a whole bunch of stuff

I noticed that tiny-js(https://github.com/gfwilliams/tiny-js) was also developed by you, and I also found many problems with it. There are many developers using tiny-js, I hope you can take the time to check it out.

gfwilliams commented on Apr 24, 2020 Member

I did notice that - you filed a *lot* of issues in it. Are you basing the usage on GitHub stars, or something else?

TinyJS was always a bit rough and ready so realistically I didn't think anyone was really using it.

rain6851 commented on Apr 24, 2020 Author

> I did notice that - you filed a *lot* of issues in it. Are you basing the usage on GitHub stars, or something else?
>
> TinyJS was always a bit rough and ready so realistically I didn't think anyone was really using it.

I am basing the usage on GitHub stars.

gfwilliams added a commit that referenced this issue on Apr 27, 2020

`Fix 1-byte overflow when using UDP (#1799)` ··· ✕ `c104311`

gfwilliams closed this as completed on May 27, 2020

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

No branches or pull requests

2 participants