New issue Jump to bottom

Memory leak when parsing a protobuf message with duplicate fields #615

Closed niooss-ledger opened this issue on Nov 24, 2020 · 5 comments

Labels Component-Decoder FixedInGit Priority-High Type-Defect

niooss-ledger commented on Nov 24, 2020

Hello,

While fuzzing a project that relies on Nanopb to parse (untrusted) user input, I found a memory leak which is triggered by sending some message where fields are duplicated.

Steps to reproduce the issue

In order to test this memleak on several versions of Nanopb (and several Linux distributions), I have written the following script:

```
#!/bin/sh
# Reproduce a memory leak issue in nanopb parser
#
# Dependencies on Debian: sudo apt install clang git protobuf-compiler python3 python3-protobuf
set -e -x

# Clone nanopb
if ! [ -d nanopb ] ; then
    git clone https://github.com/nanopb/nanopb
fi

# Create a protobuf file for some message with a header
cat > mypackage.proto << EOF
syntax = "proto3";
package mypackage;

import "nanopb.proto";

message HeaderField {
  bytes mydata = 1 [(nanopb).type = FT_POINTER];
}

message Header {
  option (nanopb_msgopt).anonymous_oneof = true;
  oneof one {
    HeaderField field = 1;
  }
}

message MessageWithHeader {
  Header head = 1;
}
EOF

# Create a fuzzer on this message
cat > fuzz_decode_message.c << EOF
#include <stdint.h>
#include <stdio.h>

#include <pb_decode.h>
#include "mypackage.pb.h"

int LLVMFuzzerTestOneInput(const uint8_t *data, size_t size) {
    mypackage_MessageWithHeader req = {};

    pb_istream_t is = pb_istream_from_buffer(data, size);
    if (!pb_decode(&is, mypackage_MessageWithHeader_fields, &req)) {
        printf("Failed to decode input: %s\n", PB_GET_ERROR(&is));
        return 0;
    }
    printf("Parsing ok, req.head.which_one = %u\n", req.head.which_one);
    pb_release(mypackage_MessageWithHeader_fields, &req);
    return 0;
}
EOF

# Compile the .proto and the fuzzer
protoc \
    -Inanopb/generator \
    -Inanopb/generator/proto \
    -I. \
    --plugin=protoc-gen-nanopb=nanopb/generator/protoc-gen-nanopb \
    --nanopb_opt= \
    --nanopb_out=. \
    mypackage.proto

clang -g -ggdb -O1 -fsanitize=fuzzer,address,undefined \
    -Wall -Wextra -Inanopb -DPB_ENABLE_MALLOC -DPB_FIELD_32BIT \
    -o fuzz_decode_message.out \
    fuzz_decode_message.c mypackage.pb.c nanopb/pb_decode.c nanopb/pb_common.c

# Run on a test case that leaks some bytes
python3 -c 'import sys;sys.stdout.buffer.write(bytes.fromhex("0a06 0a020a00 0a00"))' > memleak_message
./fuzz_decode_message.out memleak_message
```

What happens?

On a up-to-date Debian 10 machine, this leads to the following output:

```
./fuzz_decode_message.out: Running 1 inputs 1 time(s) each.
Running: memleak_message
Parsing ok, req.head.which_one = 1
Parsing ok, req.head.which_one = 1

=================================================================
==3937==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 4 byte(s) in 1 object(s) allocated from:
    #0 0x4f25a2 in realloc (/fuzz_decode_message.out+0x4f25a2)
    #1 0x536a80 in allocate_field /nanopb/pb_decode.c:581:11
    #2 0x533f3a in pb_dec_bytes /nanopb/pb_decode.c:1479:14
    #3 0x52ed88 in decode_pointer_field /nanopb/pb_decode.c
    #4 0x525632 in pb_decode_inner /nanopb/pb_decode.c:1083:14
    #5 0x5359cd in pb_dec_submessage /nanopb/pb_decode.c:1589:18
    #6 0x52d008 in decode_static_field /nanopb/pb_decode.c:532:20
    #7 0x525632 in pb_decode_inner /nanopb/pb_decode.c:1083:14
    #8 0x5359cd in pb_dec_submessage /nanopb/pb_decode.c:1589:18
    #9 0x52cea9 in decode_static_field /nanopb/pb_decode.c
    #10 0x525632 in pb_decode_inner /nanopb/pb_decode.c:1083:14
    #11 0x526c24 in pb_decode /nanopb/pb_decode.c:1159:14
    #12 0x52143d in LLVMFuzzerTestOneInput /fuzz_decode_message.c:11:10
    #13 0x42edfa in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long) (/fuzz_decode_message.out+0x42edfa)
    #14 0x422003 in fuzzer::RunOneTest(fuzzer::Fuzzer*, char const*, unsigned long) (/fuzz_decode_message.out+0x422003)
    #15 0x426b31 in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const*, unsigned long)) (/fuzz_decode_message.out+0x426b31)
    #16 0x44a3f2 in main (/fuzz_decode_message.out+0x44a3f2)
    #17 0x7fd6f93e609a in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x2409a)

SUMMARY: AddressSanitizer: 4 byte(s) leaked in 1 allocation(s).

INFO: a leak has been found in the initial corpus.
```

With my program, `0a06 0a020a00 0a00` leaks 4 bytes, `0a0a 0a020a00 0a020a00 0a00` leaks 8 bytes, etc.

What should happen?

I believe that parsing untrusted input should not leak allocated memory. You might disagree with this belief, in which case it would be nice to indicate in https://github.com/nanopb/nanopb/security/policy that Nanopb may leak memory when parsing untrusted data which was maliciously crafted.

PetteriAimonen commented on Nov 24, 2020 Member

I agree with you that parsing untrusted input shouldn't leak memory, and the fuzztest included in nanopb tries to verify that also.

It appears that your test case is hitting some situation that is not covered by the fuzztest. I'll have to take a look.

PetteriAimonen added **Component-Decoder** **Priority-High** Type-Defect labels on Nov 24, 2020

PetteriAimonen added a commit that referenced this issue on Nov 24, 2020

`Fix memory leak with oneofs and PB_ENABLE_MALLOC (#615)` … `edf6dcb`

PetteriAimonen commented on Nov 24, 2020 Member

Looks like the current test cases did not find this because they contained only a fully static fields inside a static submessage inside oneof, and pointer fields inside a pointer submessage inside oneof. And the bug is only triggered by pointer fields inside a static submessage inside oneof.

There is a preliminary fix on git now, but test cases must still be expanded to cover this. Something like #143 would be nice to help finding cases like this, but I'm not sure how to go about actually implementing that.

PetteriAimonen added a commit that referenced this issue on Nov 25, 2020

`Expand mem_release testcase to cover submessage merge (#615)` … `5ce89e0`

PetteriAimonen added a commit that referenced this issue on Nov 25, 2020

`Include a static submessage in alltypes pointer test case.` … `006a18b`

PetteriAimonen added the FixedInGit label on Nov 25, 2020

niooss-ledger commented on Nov 25, 2020 Author

Thanks for your quick reply. I confirm your patches fix the memory leak in the project that I am fuzzing (I updated Nanopb to `master` branch, which includes `edf6dcb`).

I am not familiar enough with the project to help adding what would be necessary to generate `.proto` files by fuzzing which would have detected this issue (like what is described in #143) but I will continue to fuzz some projects that use Nanopb and report other issues that I might find.

For this issue, do you know when a release will include a fix? (In a few weeks/months/...?). Also, is there any plan regarding back-porting the fix to branch `maintenance_0.3` ?

PetteriAimonen commented on Nov 25, 2020 Member

Yeah, I'm working on making a release, you can expect it today or tomorrow. And yeah, I will backport the fix and the test to 0.3 also.

PetteriAimonen added a commit that referenced this issue on Nov 25, 2020

`Expand mem_release testcase to cover submessage merge (#615)` … `d9d5dfd`

PetteriAimonen added a commit that referenced this issue on Nov 25, 2020

`Fix memory leak with oneofs and PB_ENABLE_MALLOC (#615)` ··· 4fe2359

PetteriAimonen commented on Nov 25, 2020 Member

Fix is now released in 0.4.4 and 0.3.9.7.

PetteriAimonen closed this as completed on Nov 25, 2020

PetteriAimonen added a commit that referenced this issue on Nov 25, 2020

`Documentation: clarify security model on dynamic allocations (#615)` 32d9a1e

jszumski mentioned this issue on Dec 3, 2020

Remediate CVE-2020-26243 by updating to nanopb 0.3.9.7 or higher firebase/firebase-ios-sdk#7090

Closed

Assignees

No one assigned

Labels

Component-Decoder FixedInGit Priority-High Type-Defect

Projects

None yet

Milestone

No milestone

Development

No branches or pull requests

2 participants