RootMyTV / **RootMyTV.github.io** Public

RootMyTV is a user-friendly exploit for rooting/jailbreaking LG webOS smart TVs.

rootmy.tv

MIT License

120 stars **6** forks

Star Notifications

Code Issues 4 Pull requests Actions Projects Wiki ···

main

Go to file

Informatic index: we are public already... ··· ✓ 11 days ago **49**

View code

README.md

RootMy.TV

RootMyTV is a user-friendly exploit for rooting/jailbreaking LG webOS smart TVs.

It bootstraps the installation of the webOS Homebrew Channel, and allows it to run with elevated privileges. The Homebrew Channel is a community-developed open source app, that makes it easier to develop and install 3rd party software. Find out more about it here.

If you want the full details of how the exploit works, skip ahead to our writeup.

Is my TV vulnerable?

At the time of writing (2021-05-15), all webOS versions between 3.4 and 6.0 we tested (TVs released between mid-2017 and early-2021) are supported by this exploit chain. Note: this versioning refers to the "webOS TV Version" field in the settings menu, *not* the "Software Version" field.

If you want to protect your TV against remote exploitation, please see the relevant section of our writeup and/or await an update from LG.

Usage Instructions

Step Zero (disclaimer): Be aware of the risks. Rooting your TV is (unfortunately) not supported by LG, and although we've done our best to minimise the risk of damage, we cannot make any guarantees. This may void your warranty.

1. Make sure the "LG Connect Apps" feature is enabled. It seems to be enabled by default on webOS 4.0+. For older models, follow LG's instructions.
2. (Optional but recommended) If you have LG's Developer Mode app installed, uninstall it. You won't be able to use it after running the exploit, and its functionality is replaced by the Homebrew Channel.
3. Open the TV's web browser app and navigate to https://rootmy.tv
4. "Slide to root" using a Magic Remote or press button "5" on your remote.
5. Accept the security prompt.
6. The exploit will proceed automatically. The TV will reboot itself once during this process, and optionally a second time to finalize the installation of the Homebrew Channel. On-screen notifications will indicate the exploit's progress. Occasionally, the TV may turn off instead of rebooting - if this happens, just turn the TV back on again.

Your TV should now have Homebrew Channel app installed, and an unauthenticated(!) root telnet service exposed.

For exploiting broken TVs, check out the information here.

Post-Installation Advice (IMPORTANT!)

1. For security reasons, it is **highly recommended** to disable Telnet, and enable SSH Server with public key authentication (Homebrew Channel → Settings → SSH Server). You will need to manually copy your SSH Public Key over to `/home/root/.ssh/authorized_keys` on the TV.

   GitHub user registered keys can be installed using the following snippet:

```
mkdir -p ~/.ssh && curl https://github.com/USERNAME.keys > ~/.ssh/authori
```

2. Don't update your TV. While updates are technically possible, if LG patches the exploit, you might end up "locked out" and unable to re-root your TV if you somehow lose access. We also can't predict how future updates will affect our techniques used to elevate and operate the Homebrew Channel app. "Block system updates" option in Homebrew Channel will disable firmware update checks.

3. Don't Install, Uninstall, or Update LG's "Developer Mode" app. Doing so will overwrite or remove the startup script used to bootstrap the jailbreak. It is advisable to remove "Developer Mode" app before rooting. SSH service exposed by Homebrew Channel is compatible with webOS SDK tooling.

Troubleshooting

In case of any problems join the OpenLGTV Discord server and ask for help on `#rootmytv` channel, or file a GitHub issue.

Before asking for support, please consult our Troubleshooting guide.

Research Summary and Timeline

RootMyTV is a chain of exploits. The discovery and development of these exploits has been a collaborative effort, with direct and indirect contributions from multiple researchers.

On October 05, 2020, Andreas Lindh reported a root file overwrite vulnerability to LG. On February 03, 2021, Andreas published his findings, demonstrating a local root exploit against the webOS Emulator (a part of LG's development SDK). LG had boldly claimed that this issue did not affect their devices, and that they were going to patch their emulator.

On February 15th, 2021, David Buchanan reported a vulnerability in LG's "ThinQ login" app, which allowed the app to be hijacked via a specific sequence of user inputs, allowing an attacker to call privileged APIs. On March 23rd 2021, David published a proof-of-concept exploit, which enabled users to gain root privileges on their LG smart TVs. This was made possible by combining it with the local root vulnerability previously reported by Andreas (Yes, the same one that LG said did not affect their devices!).

Around March 28th 2021, Piotr Dobrowolski discovered a similar vulnerability in the "Social login" app, which is present across a wider range of webOS versions. More importantly, this exploit could be easily triggered over the local network, using SSAP (details below), making it much more reliable and user-friendly.

At time of writing, the code in this repo is the combined work of David Buchanan (Web design, initial PoC exploit) and Piotr Dobrowolski (Improved "v2" exploit implementation, and writeup).

We would like to thank:

- Andreas Lindh for publishing his webOS research.
- The wider webOS community, particularly the XDA forums and the OpenLGTV discord.
- All the contributors (present and future) to the Homebrew Channel, and development of other homebrew apps and software.
- LG, for patching symptoms of bugs rather than underlying causes...

The Technical Details

Background

webOS, as the name suggests, is a Smart TV operating system mostly based on web technologies. Applications, both system and external are either run in a stripped down Chromium-based web browser ("WebAppMgr") or in Qt QML runtime. Almost all system and external applications run in chroot-based jails as an additional security layer.

"Web apps", outside of standard web technologies, also get access to an API for communicating with "Luna Service Bus". This is a bus, similar to D-Bus, used to exchange messages and provide various services across different security domains. Bus clients can expose some RPC methods to other applications (identified by URIs `luna://service-name/prefix-maybe/method-name`) which accept JSON object message as their call parameters, and then can return one or many messages. (depending on the call being "subscribable" or not)

While Luna bus seems to have extensive ACL handling, considering the history of webOS IP transfers, seems like not many engineers fully understand its capabilities. Part of the bus is marked as "private", which is only accessible by certain system applications, while most of the other calls are "public" and can be accessed by all apps.

Unexpectedly, one of the internal services exposed on a bus is "LunaDownloadMgr" which provides a convenient API for file download, progress tracking, etc... Said service has been researched in the past and an identity confusion bug leading to an arbitrary unjailed root file write vulnerability has been publicly documented.

This in and of itself was not very helpful in production hardware, thus we needed to find a way of calling an arbitrary Luna service from an application with a `com.webos.` / `com.palm.` / `com.lge.` application ID.

Step #0 - Getting in (index.html)

In order to gain initial programmatic control of the TV GUI, an interface called "LG Connect Apps" can be used. Its protocol, called "SSAP" (Simple Service Access Protocol), is a simple websocket-based RPC mechanism that can be used to indirectly interact with Luna Service bus, and has been extensively documented in various home-automation related contexts. We use that to launch a vulnerable system application which is not easily accessible with normal user interaction.

Step #0.1 - Escaping the origins

SSAP API is meant to be used from an external mobile app. For the sake of simplicity, though, we wanted to serve our exploit as a web page. This lead us to notice that, understandably, the SSAP server explicitly rejects any connections from (plaintext) HTTP origins. However, there was an additional exception to that rule, and seemingly the authors wanted to allow `file://` origins, which present themselves to the server as `null`. Turns out there's one other origin that can be used that is also reprted as `null`, and that is `data:` URIs.

In order to exploit this, we've created a minimal WebSocket API proxy implementation that opens a hidden iframe with a javascript payload (which is now running in a `data:`/`null` origin) and exchanges the messages with the main browser frame. This has been released as a separate library.

Step #0.2 - General Data Protocol Redirection

There's a minor problem with establishing the connection with the SSAP websocket server. While we all believe in utter chaos, we don't feel very comfortable with serving our exploit over plaintext HTTP, which would be the only way of avoiding Mixed Content prevention policies. (by default, https origins are not allowed to communicate with plaintext http endpoints)

While some newer Chromium versions do allow Mixed Content communication with `localhost`, that was not the case when Chromium 38 was released (used in webOS 3.x). Thankfully, it seems like the system browser on webOS 3.x is also vulnerable to something that has been considered a security issue in most browsers for a while now - navigation to `data:` URIs. Thus, when applicable, our exploits attempts to open itself as a `data:` base64-encoded URI. This makes our browser no longer consider the origin being secure, and we can again access the plain-http WebSocket server.

Mitigation note

An observant reader may have noticed that the service we use is meant to be used remotely. While the connection itself needs a confirmation using a remote **we highly recommend to disable LG Connect Apps functionality** in order to prevent remote exploitation. However, this option seems to only be present on webOS versions older than webOS 4.x - in such cases the only solutions are to either **keep the TV on a separate network**, or disable SSAP service manually using the following command after rooting:

```
luna-send -n 1 'palm://com.webos.settingsservice/setSystemSettings' '{"catego
```

Step #1 - Social login escape (stage1.html)

Having some initial programmatic control of the TV via SSAP, we can execute any application present on the TV. All cross-application launches can contain an extra JSON object called `launchParams`. This is used to eg. open a system browser with specific site open, or launch a predetermined YouTube video. Turns out this functionality is also used to select which social website to use in `com.webos.app.facebooklogin`, which is the older sibling of `com.webos.app.iot-thirdparty-login` used in initial exploit, present on all webOS versions up until (at least) 3.x.

When launching social login via LG Account Management, this application accepts an argument called `server`. This turns out to be a part of URL that "web app" browser is navigated to. Thus, using a properly prepared `launchParams` we are able to open an arbitrary web page (with the only requirement being that it's served over `https`) running as a system app that is considered by `LunaDownloadMgr` a "system" app.

Step #2 - Download All The Things (stage2.html)

Since we are already running as a system application, we can download files (securely over https!) into arbitrary unjailed filesystem locations as root.

We use that to download following files:

- `stage3.sh` → `/media/cryptofs/apps/usr/palm/services/com.palmdts.devmode.service/start-devmode.sh` - this is the script executed at startup by `/etc/init/devmode.conf` as root, in order to run developer mode jailed SSH daemon.
- `hbchannel.ipk` → `/media/internal/downloads/hbchannel.ipk` - since our end goal is intalling the Homebrew Channel app, we can also just download it during the earlier stages of an exploit and confirm it's actually downloaded.
- `devmode_enabled` → `/var/luna/preferences/devmode_enabled` - this is the flag checked before running `start-devmode.sh` script, and is just a dummy file.

Step #3 - Homebrew Channel Deployment (stage3.sh)

`stage3.sh` script is a minimal tool that, after opening an emergency telnet shell and removing itself (in case something goes wrong and the user needs to reboot a TV - script keeps running but will no longer be executed on next startup), installs the homebrew channel app via standard devmode service calls and elevates its service to run unjailed as root as well.

Releases

No releases published

Contributors 4

DavidBuchanan314 David Buchanan

Informatic Piotr Dobrowolski

ledoge

Ruthenic Drake

Languages

● **HTML** 71.9% ● **CSS** 18.9% ● **Shell** 9.2%