Table of Contents

The Blog

What is wrong with the 2023 League worlds opening ceremony?

The original league worlds opening ceremony video from this year has very many technical quirks. Here is what I noticed and spent a a few (prolly a dozen) hours fixing.

- The audio on the YouTube upload is unexplicably in mono and sounds notably worse than the Twitch livestream which was in stereo.
- Likely destructive interference when collapsing stereo to mono?
- The whole 'scripted' section of the opening is actually running at 29.97 fps, and interpolated to 60 but in the worst way.
- Half of their cameras are running at 29.97 which means every frame is duplicated for the stream which is 59.94 - this on it's own is fine. Most of the side-angle cameras are 29.97.
- The other half of the cameras are also running at 29.97, but are interpolated to 59.94 THROUGH FRAME BLENDING?!?
- This means that every other frame is literally a 50/50 blend of the two frames before it. This is what causes that motion-blur like feel, and you can instantly tell which cameras they are on.
- It might be the cameras that also do AR are doing blending? unsure.
- In fact, the main stage camera and the cable cam blend on opposite frames - eg: one has good even frames, blended off, the other is vice versa. This might be because of an internal delay? I'm not sure. Don't get me wrong 1080p29.97 is fine, although odd since sports is usually 720p60 or 1080i60.
- The version on YouTube took a video that had TV black levels, mapped it to PC, then was converted to TV again, resulting in crushed black and white levels losing detail
- The version on Twitch has the audio and video about 100 ms out of sync, this is most noticeable with the fireworks at the end of gods not being in line with music. Interestingly, the audio and video ARE synced correctly on YouTube version.
- TV (and most video formats) uses 16-235 to express black-white instead of 0-255, whereas PC is 0-255. So, if you take a 16-235 feed, map it to 0-255, then crop it into 16-235 again, you effectively increase contrast and reduce your dynamic range, that is what seems to have happened on YouTube.
- Another odd quirk: some of the camera angles are different between the YT and Twitch versions? Either these were cut after the fact, or it was filmed weird? Not sure, I can upload a side by side comparison if anyone is curious. Notable on some audience and a few side angle shots.
- Also if you watch the side by side, depending on if you sync up the video or audio feeds, it almost looks like there are 2 video systems, and depending on which you sync up on, some transitions happen 100 ms before or after the other, with the other half being synced. I think the AR stuff is probably separate?
- About that frame blending, there are 4k60 audience camera recordings on YouTube, and if you watch those carefully, you'll notice the camera feed going to the video walls also does frame blending. This tells me that whatever system video was going into first was doing the blended frames, but again, only on half of the cameras.

So, to fix this

- I took the stream from Twitch, split the audio and video. For the video, I couldn't simply drop all odd or even frames because their different cameras interpolate different sets of frames.
- From my cut, the “main” stage camera had odd frames correct, and even frames blended, and the cable cam that is further away ad all even frames correct and odd frames blended.
- So I spent 3 hours clipping each camera angle, and dropping either the odd or even frames accordingly.
- I then re-synced the audio with that 100ms offset. This left me with a 29.97 fps video. That is my “cleaned up” version that I might upload, but, since the original was 60 fps, I wanted to interpolate it, but properly, not through frame blending.
- So I ran the whole thing through an AI video interpolation tool overnight on my GPU, and there you have it, 4k60 (technically 59.94). I'll add the 29.97 maybe as well.

Stuff I did not fix:

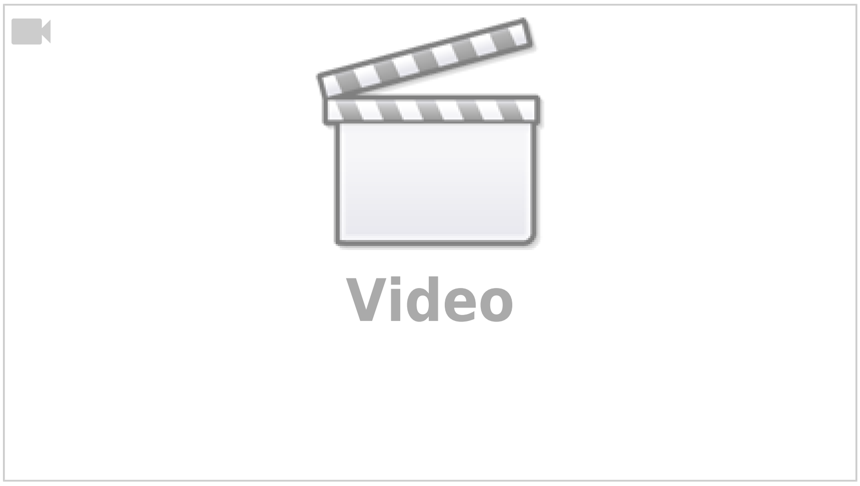
- The yellow jacket intro video is filmed in 24 fps, and brought up to 30, but not in a very consistent way, so it's staying as is.
- I can't change mic volumes, there isn't a lot, but you can pretty easily tell which parts are lip-synced anyway (listen for the reverb different, or when vocals sounds like they are stereo - eg: first 10s of vocals in gods sounds live, most of the rest is off the tape).
- I also can't fix their broken shadows on their AR characters.

Note: I do not own the music or video, those are from Riot Games. I'm simply uploading this in case anyone else wanted to watch it without the bizarre quirks of the version on YouTube.

If anyone from the Riot Games technical team is watching, I'm genuinely curious about how their production setup works and I would love to chat. (Or if you know someone who works with the technical team)

This is not meant to be a dunk on the performance, overall it was well put together, these are just 'quirks' I saw and couldn't unsee.

Here is the fixed version



2023-12-20 07:36 · Tony

Thinkpad T14 Gen2 AMD eDP Screen/Panel Upgrade

If you read Thinkpad T14 Gen2 AMD, I hated the screen they shipped with the T14, and thought it was a disgrace for a “premium” laptop.
Well, it annoyed me so much that I started to research how to upgrade the screen of the laptop.

Original Panel

The original panel is a was an Innolux N14HCA-EAE.
It is IPS-type, <250 nits, and has garbage colours.

Since it is 1080p non-touch, replacing it with another 1080p panel isn't too hard.
If you had a 4k screen, you would need a new eDP cable. If touch, replacing the digitizer is another step.

Look for the Thinkpad maintenance guide for removing an old screen. Follow it like Lego.
This is an irreversible process. Save the bezel. It's supposed to be replaced, good luck finding a replacement. Pry it slowly.
The original screen is held by double sided tape. I didn't replace mine, you maybe should - again, good luck.

New Panel

I bought the Innolux N140HCG-GQ2.
It's low power, 400 nit, 90+% sRGB, IPS-like, 1080. Much better.
No new cable needed, drop in replacement basically. Comparison sites label it the best of the 3/4 “low power” display options.
You can find via AliExpress and hope it's not a fake.

Re-use existing tape and bezel when reassembling, clip display properly.

Worth it?

100%. The brightness and colour gamut are very much noticable and appreciated.
It makes me actually enjoy looking at the laptop screen.

If you have the patience and courage, go for it! LMK how it went!

2023-07-02 15:42 · Tony

CVE-2022-0847 - Dirty Pipe Vulnerability in Linux

Introduction

The Linux Dirty Pipe vulnerability, also known as CVE-2022-0847 is major a vulnerability first discovered near the end of February 2022 which affects Linux kernel versions 5.8 and above. The Dirty Pipe vulnerability is considered severe since there aren't workarounds (besides upgrading to a patched version of the Linux kernel) and it allows local privilege escalation allowing users to write to files they don't have write access to (such as /etc/passwd), which then allows them to run commands as root or drop to a root shell.

Before I go any further, if you are reading this post from a computer running Linux (or GNU/Linux), make sure you have updated your kernel to 5.16.11, 5.26.36 or 5.10.102 to make sure you're protected against this vulnerability!

This vulnerability was first discovered last month when investigating an older bug report regarding corrupted files. In particular, there were reports of corrupted CRC checksums at the end of zip files in a particular use case involving splices and pipes.

Some background information

There are a few background elements that I think are useful to understand this vulnerability and how it works: Pipes, pages, the page cache and the splice system call on Linux.una20

First: Pipes. Pipes in Linux are a unidirectional, inter-process communication method provided by the kernel. Pipes are commonly used to pass information between 2 applications. The output of the first application is saved to the pipe, which is then read by the second application. Pipes can be used both in the shell using '|' to redirect the output of one program to the input of the next (such as if you do cat foo | grep a | cut -c 2- | uniq | sort | tee bar), or within programs, which is more flexible and powerful. Under the hood, pipes are treated and behave just like files, which is part of the Unix philosophy. However, unlike files on secondary storage, the data in pipes is stored in a buffer (that works like a ring using pages) instead of on the disk, where data from the first application is added to the buffer, and the second application reads from it. In other words, it's a handy way to send data from one application to another, where one pushes data into the pipe, and the other pulls it from the pipe.

Next: Pages. As discussed in the earlier modules of the course, pages are blocks of virtual memory. Pages are generally the smallest unit of data used for virtual memory. These pages are what will contain files, data, code, so on, and can be loaded from the secondary disk (SSD/HDD), or from system memory. On Linux based systems, the page size is typically 4 KiB (4096 bytes). (You can check the page size for yourself by running 'getconf PAGESIZE')

To speed up I/O operations when a user is frequently working with certain pages, Linux maintains a page cache, which will essentially cache the commonly used items from secondary memory (SSD/HDD) onto primary RAM. This means that if you're continually loading a part of a file, instead of waiting for a hard drive each time, it can be cached and retrieved from the cache in memory. This cache is usually in unused parts of memory, and can be easily cleared if an application needs to actually use the memory. Pages in the page cache that are modified while in memory (such as by a write to the file) will mark the page as 'dirty', which have to be written back to the disk to save/persist those changes.

Finally, the splice() system call. Splice is a system call that moved data between files on the disk and pipes without having to load and go through the userspace. Generally, splice can work by just remapping pages, which makes it fast. This means that instead of passing the page around, it passes page references, without having to do the copy. You can splice a file to a pipe, then splice back from the pipe into another file without dropping to userspace, which makes this both fast and secure.

What are the causes?

The dirty pipe vulnerability takes advantage of a pipe and a splice call. This started as a bug that was introduced with a commit which refactored the pipe buffer code, changing how 'mergeable' checks were done.

As mentioned in the section above, all data, including the data temporarily stored in a pipe lives in a page, which can get cached into memory. The first write into a pipe allocates a 4 KiB page. Then, this page is used for writes until it fills up, at which point a new page is allocated. However, if you use the splice system call, the kernel first loads the data from the file (source) into the page cache, then, creates a 'pipe_buffer' struct (the destination pipe) which points to the page cache. This is a zero-copy, since it essentially just copies the reference to the page in the page cache. However, even if the page isn't full, it can't be appended to, since the page is owned by the page cache, not the pipe.

The checks to ensure that data isn't appended to page in the page cache have changed over time, but the latest revision with Linux 5.8 turned a bug into a critical issue. Since kernel 4.9, there has been a bug causing the flags member of the pipe_buffer struct to be missing, however, the new critical issue allows a user to inject PIPE_BUF_FLAG_CAN_MERGE into the page cache reference, which then allows the user to overwrite data in the page cache! Once the PIPE_BUF_FLAG_CAN_MERGE flag is injected into the reference, writing new specially crafted data into the pipe that was originally loaded with the splice causes the kernel to append this data to the end of the page in the page cache, instead of creating a new page.

Since the splice operation loaded the contents of a file into the pipe, by placing the pages for the file in the page cache, if we can write to the page cache, we can effectively write to the version of the file that will be used by the OS. This is critical, since you only need read-access in order to load the contents from a file into a pipe. However, as seen here, by setting the flag, we can write to the page in the page cache. This means that any user can write into files they would normally only have read access to. Essentially, the kernel is tricked into thinking this is a normal page for the pipe that is safe to write to, however, it's actually the contents of other files. Yikes.

Now, there is one aspect of this to consider. By using this vulnerability, you can write data into the page cache, which is loaded with the contents of another file. However, since the kernel is unaware of this change, the page in the cache is not marked as dirty. This means that eventually, either on system restart, or if the kernel needs to reclaim the memory, the page in the cache will be dropped. This is why this bug was hard to spot in the wild, since in most cases, it would be gone by the time the system restarted. On the other hand, since the data in the page cache isn't written to the disk, and lasts for a considerable amount of time, this provides a stealthy attack surface that doesn't leave a trace on the disk.

Why is it important?

This vulnerability is critically important, since it can be exploited to achieve local privilege escalation. This vulnerability allows any unprivileged user to write to pages in the page-cache which correspond to files they only have read access to. Since the kernel will use the page cache to load a file if it is cached, this effectively allows the user to overwrite data into any file they have read access to, with some limitations.

This can be exploited to get root access (or a root shell) through several means. For example, through a few steps, one can simply clear the root password from /etc/passwd, add SSH keys, or change data used by root processes. The simplest is likely by disabling the password for the root user. While normal users can't access /etc/shadow, by removing the x in /etc/passwd, you can remove the lookup to /etc/shadow, and allow password-less login to the root user. For those who are curious, there is a proof-of-concept exploit available on the original dirty pipe vulnerability page that allows writes into read-only files. Once an attacker has root access on your system, there's little they can't do.

A system with this vulnerability can be exploited by attackers for any malicious purpose following a few steps. First, the user creates a pipe. This is a pipe which they must have write access to. Next, you fill the pipe with random or arbitrary data. The contents don't matter, it just needs to fill the pipe so that the PIPE_BUF_FLAG_CAN_MERGE which is later incorrectly checked will get set in all of the pages and entries used by the pipe. Next, you empty the pipe, by reading the data from it. This leaves the flag set in all instances of the pipe_buffer struct in the ring.

For step 4, you need to make a splice system call for the file you want to modify, opened with O_RDONLY (read only), and splice it into the pipe just up to the target offset. The target offset is the location in the file where you want to write your new data to and make your changes. For example, this could be the location of the root user password. Next, you write the data you want to replace in the file into the pipe. As discussed in the last step, instead of creating a new anonymous pipe_buffer struck with new pages, the new data added to the pipe will begin to replace the data in the page from the read-only file (which is in the page cache) because the PIPE_BUF_FLAG_CAN_MERGE flag is set. Recall that that flag would normally only be set if this were a normal pipe, but remained set.

This exploit was particular interesting (and terrifying) for myself since I had my secondary computer running an earlier build of Linux 5.16, which was vulnerable, and I was able to test both the small code by the original author, as well as their proof-of-concept exploit which both worked. While not trivial, it is a vulnerability that is relatively simple to exploit. While there was somewhat similar exploit named Dirty Cow from a few years ago, this is simpler to exploit.

There are however a few limitations to these exploits, although they are all simple to overcome once you gain root access or a root shell. First, you need to have read access to a file. Without read access, you won't be able to use the splice call. Next, the offset that you want to overwrite can't be on a page boundary (ie: can't be a multiple of 4096). This is because you need to splice at least 1 byte from the target file into the pipe. Next, you are limited to overwriting 1 page (4096 bytes). While this might seem short, since you can't change large files, it's more than enough to update important files such as /etc/passwd. Finally, you can't resize the file, so the data you write can't alter the file size. Again, once you have root access, the other limitations wouldn't matter for an attacker, since they can use their root access to change any other files they wanted.

Who is affected and in what ways?

Any device running Linux 5.8 or later (without the latest patches) are affected. This includes many servers, desktop computers and laptops, as well as many recent mobile devices running Android. In particular, systems running many major distributions from the past 2 years without the most recent updates are vulnerable. This includes Debian 11 (bullseye), Fedora 33 and up, Ubuntu 20.04 and up, as well as many others including Arch, Gentoo, OpenSUSE, as well as their derivatives. This constitutes a good portion of server and desktop systems running recent installations of GNU/Linux.

Although it is a local vulnerability, meaning that it doesn't add a new vector for an outside attacker, if an attacker has any access to a user on the system, be it physical or digital such as over SSH, Telnet or VNC, they can exploit this vulnerability to gain root access.

Equally important is the use of the Linux kernel in the Android mobile operating system. Within a few days of the discovery of the original vulnerability, the bug was reproduced on a Google Pixel 6, and reported to the Android Security Team. If the bug is reproduced, then it means that there will also be exploits targeting these devices. Of course, this impacts more devices than just the Pixel 6. For example, the Samsung Galaxy S22 series are all based on Android 12, running Linux kernel 5.10.43, which is vulnerable. Android versions prior to Android 11 used Linux kernel versions 4.4, 4.9, 4.14, 4.19 or 5.4, however, starting with Android 12, support for Linux 5.10 was added, with Android T (the experimental branch) also using 5.10. This means that many flagship phones are going to ship with a build of Linux 5.10. Given the poor record of Android manufacturers issuing software updates, this can end up affecting a good number of recent phones that were released.

How can similar problems be prevented in the future?

Given that this bug was introduced due to programmer error when refactoring a section of the code, it's harder to provide many meaningful suggestions. A solution that is frequently presented is to have a more stringent code-review process, with more developers or a stronger emphasis on security. However, patches going into the Linux kernel already undergo a thorough code-review from multiple developers. While there are also suggestions to avoid refactoring some code since it can introduce bugs such as this one, refactoring is also important in order to clean up the code and keep it consistent.

Some mistakes like this could be caught in part by using analysis tools. For example, the basis of this vulnerability was originally a bug with uninitialized flags for the struct. This type of mistake such as uninitialized variables in structs could be caught if the team added corresponding checks. However, this of course comes with the downside that the developers might also get bogged down with tons of false positive reports. Some have also suggested switching to a safer language such as Rust, but in cases like this where it's very specific behavior in specific calls and specific checks, which don't necessarily break the safety of the language, it's debatable it would have caught a mistake such as this one.

We have also seen that having a simpler codebase makes auditing much simpler while also making it both faster and safer since it becomes easier to stop mistakes. For example, the Wireguard VPN protocol which was written to be a faster, safer VPN protocol was welcomed into the Linux kernel in that past years since it's so much simpler then competing protocols such as OpenVPN (which is based on the comparatively massive OpenSSL and TLS libraries). However, a kernel is much more sophisticated than

VPN protocols, and given that Linux is a monolithic kernel, it is much harder to just have simpler code - although there are projects such as Redox OS which are both supposed to be simpler, using a microkernel architecture, as well as being written in the safer Rust language.

Overall, this is one of the aspects where open source shines. We can always have more developers looking at and through the code, and it allows users such as Max Kellermann (the original author of the vulnerability) to not only find bugs such as this one, but also patch them in a timely manner.

If such a problem happened, how can we mitigate it?

This vulnerability was reported professionally by the security researcher, who alerted the Linux kernel security team within a day of his discovery of the vulnerability, along with a patch. This was quickly sent to the Linux Kernel Mailing List the next day, where it was merged into the next release within a few days. Overall this problem fixed very quickly by the Linux developers, with patched versions of the kernel out within a week of the discovery of the original vulnerability. This is excellent, and it shows that security issues such as this one are taken very seriously by the developers who can work through these issues and patch them in an effective manner.

Once again, this is a good example of how the open source software community is very good at responding to issues like this once they are found with everyone working together to get it patched and ensure systems get updated. This vulnerability was fixed for the release of versions 5.16.11, 5.15.25, and 5.10.102. Of these, 5.16.11 is the latest stable release that distributions like Fedora and Arch use, with 5.15 and 5.10 being LTS releases.

Once the new versions of the kernel were released, all of the distributions mentioned in my section above have either upgraded their kernel to include the patch for this vulnerability, or, backported the patch for this vulnerability into their kernel versions. This is a good example of the author, kernel developer, as well as distribution maintainers working together to ensure security issues like this get patched.

As I mentioned in the start of this post, the mitigation for this vulnerability for both desktop and server systems is to update your kernel! Whether you're maintaining a server or your system, make sure your software is up-to-date! For servers in particular, make sure you're running a version of your distribution that is actively maintained, such as an LTS release to ensure patches like this one get backported to ensure your systems are safe.

Mobile users on the other hand will have to wait for updated versions of Android to be released for their devices, likely through security updates. We can hope that they will be as responsive as distribution maintainers are, and apply these patches and deliver new updates in a timely manner. Since this bug was introduced in Kernel 5.8 from a few years ago, and the most recent Android version to ship with a vulnerable kernel is Android 11, most devices that are vulnerable are relatively new, increasing the chance they will get patched.

Conclusion

I hope you enjoyed reading this and learned about the dirty pipe vulnerability on Linux, and how pipes, pages, the page cache and splice call work in general!

Biblography

- "CVE-2022-0847," CVE, 03-Mar-2022. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-0847. [Accessed: 14-Mar-2022].
- D. Das, "What is the dirty pipe exploit in linux and how can you fix it?," Make Use Of, 10-Mar-2022. [Online]. Available: https://www.makeuseof.com/what-is-the-dirty-pipe-exploit-in-linux-and-fix-it/. [Accessed: 14-Mar-2022].
- D. LeClair, "New dirty pipe linux vulnerability is the worst in years," How-To Geek, 08-Mar-2022. [Online]. Available: https://www.howtogeek.com/790435/new-dirty-pipe-linux-vulnerability-is-the-worst-in-years/. [Accessed: 14-Mar-2022].
- M. Kellermann, "The dirty pipe vulnerability," The Dirty Pipe Vulnerability - The Dirty Pipe Vulnerability documentation, 07-Mar-2022. [Online]. Available: https://dirtypipe.cm4all.com/. [Accessed: 14-Mar-2022].
- P. Arntz, "Linux 'dirty pipe' vulnerability gives unprivileged users root access," Malwarebytes Labs, 11-Mar-2022. [Online]. Available: https://blog.malwarebytes.com/exploits-and-vulnerabilities/2022/03/linux-dirty-pipe-vulnerability-gives-unprivileged-users-root-access/. [Accessed: 14-Mar-2022].
- S. Hazarika, "Linux kernel bug dubbed 'dirty pipe' can lead to root access, affects Android devices as well," XDA Developers, 09-Mar-2022. [Online]. Available: https://www.xda-developers.com/dirty-pipe-linux-vulnerability-root-android/. [Accessed: 14-Mar-2022].

Further reading

The original post disclosing this vulnerability has lots of other technical information, including how the vulnerability was discovered due to a problem with repeated corruption on web-server logs. https://dirtypipe.cm4all.com/

The commit that introduced this vulnerability is here: https://github.com/torvalds/linux/commit/f6dd975583bd8ce088400648fd9819e4691c8958

For those of you that want to try the proof-of-concept exploit on your unpatched devices, the source code is available here: https://packetstormsecurity.com/files/166229/Dirty-Pipe-Linux-Privilege-Escalation.html

If you have any questions or comments, feel free to leave it below!

2022-03-14 07:06 · Tony

Linux Audio Essentials

This post is also in the form of a video blog. Watch the video first!

- Youtube: https://youtu.be/HxEXMHcwtlI
- Peertube: https://peertube.tonytascioglu.com/videos/watch/4a6b4e74-a5af-4616-96f4-5d9773033152

The description below is longer then the one on YouTube, which passed the maximum 5000 character limit. The version here has all the full URLs and any extra comments I might add.

Extra Insight

In this video, I explain how audio and sound works on Linux based comptuers and systems. More specifically, I go over the point of sound hardware, kernel drivers such as OSS and ALSA and userspace sound servers such as PulseAudio, Jack and Pipewire.

Along the way, I discuss the advantages and drawbacks of the current implementations, as well as why one implementation is often favored over another. Finally, I discuss the latest-and-greatest sound server, Pipewire, what it means, and how you can benefit from the improvements.

This video is a bit rambly at times, so please stick with me, and I hope you learn something throughout and feed your curiosity. Please feel free to use the timestamps below to skip between sections!

Timestamps

(grouped by topics)

Introduction

- 00:00 - Introduction

The Hardware

- 00:18 - Basic Hardware, Inputs and Outputs
- 00:36 - Sound Cards (and what they do)
- 01:01 - Digital Audio, PCM and extra hardware

Kernel Drivers

- 01:29 - Kernel Drivers! (How to interact with hardware)
- 01:53 - OSS (Open Sound System)
- 02:12 - ALSA (Advanced Linux Sound Architecture)
- 02:46 - ALSA Limitations - hardware mixing/multiplexing

Userspace Sound Servers

- 03:54 - Pulseaudio (and sound servers)
- 04:25 - Benefits of PA - mixing and resampling

- 07:26 - Drawbacks of PA (and JACK introduction)
- 08:13 - JACK and its benefits
- 09:57 - Comparison with PA and other software

Pipewire (and ramble)

- 11:12 - Pipewire (and its benefits)
- 14:05 - Future of Pipewire
- 15:17 - Note on Bluetooth (rant)

- note: mostly fixed!

- 17:52 - Conclusion

Links (and references)

Sound Cards

- https://en.wikipedia.org/wiki/Sound_card

Check ALSA compatibility of a sound card

- https://www.alsa-project.org/wiki/Matrix:Main

DAC and ADC

- https://www.ramelectronics.net/analog-digital.aspx
- https://en.wikipedia.org/wiki/Digital-to-analog_converter

Nyquist Shannon sampling theorem

- I didn't get to it in this video, but it explains why 44.1 and 48 kHz are perfectly fine.
- More specifically, how we can perfectly reconstruct analog waves provided no aliasing and they are below the nyquist frequency.
- https://www.allaboutcircuits.com/technical-articles/nyquist-shannon-theorem-understanding-sampled-systems/

Chris Montgomery Videos

- I found these super helpful to understand digital audio and video fundamentals.
- Discusses PCM and more, and also the nyquist stuff from above in video 2.
- https://www.xiph.org/video/

- https://wiki.xiph.org/A_Digital_Media_Primer_For_Geeks_%28episode_1%29
- https://wiki.xiph.org/Digital_Show_and_Tell/Episode_02

- Also see Chris' blog while you're at it, some interesting reads:
- https://xiphmont.dreamwidth.org/

Kernel Driver Architecture

- I found this a simple overview when researching
- https://events19.linuxfoundation.org/wp-content/uploads/2017/12/Introduction-to-Linux-Kernel-Driver-Programming-Michael-Opdenacker-Bootlin-.pdf

OSS

- https://en.wikipedia.org/wiki/Open_Sound_System
- https://wiki.archlinux.org/title/Open_Sound_System

- ie: OSS wasn't bad, and had some afvantages over ALSA, but the licensing switch just prompted people to switch
- https://en.wikipedia.org/wiki/Open_Sound_System#Free,_proprietary,_free

ALSA

https://www.alsa-project.org/wiki/Main_Page

- The sound card compatibility list is above. The Gentoo and Arch wiki entries are useful.
- https://wiki.archlinux.org/title/Advanced_Linux_Sound_Architecture
- https://wiki.gentoo.org/wiki/ALSA

Sound card multiplexing

- https://en.wikipedia.org/wiki/Sound_card_mixer
- https://newbedev.com/why-do-you-need-pulseaudio

- Use a sound server. Don't do this manually https://electronics.stackexchange.com/questions/57476/how-do-i-multiplex-many-signals-into-my-sound-card

Pulseaudio

- Homepage: https://www.freedesktop.org/wiki/Software/PulseAudio/
- User docs: https://www.freedesktop.org/wiki/Software/PulseAudio/Documentation/User
- Git: https://gitlab.freedesktop.org/pulseaudio/pulseaudio

- As usual, the arch page and examples are good:
- https://wiki.archlinux.org/title/PulseAudio
- https://wiki.archlinux.org/title/PulseAudio/Examples

Jack

- Homepage: https://jackaudio.org/
- Jack1 git: https://github.com/jackaudio/jack1
- Jack2 git: https://github.com/jackaudio/jack2
- Wiki (and tools using Jack) https://github.com/jackaudio/jackaudio.github.com/wiki
- Archwiki: https://wiki.archlinux.org/title/JACK_Audio_Connection_Kit

Pipewire

- Hoempage: https://pipewire.org/#about
- Neat demo and features, and other benefits discussed on hackaday here: https://hackaday.com/2021/06/23/pipewire-the-newest-audio-kid-on-the-linux-block/
- Archwiki as always: https://wiki.archlinux.org/title/PipeWire
- Wiki - contains useful config parameters for pulse and jack: https://gitlab.freedesktop.org/pipewire/pipewire/-/wikis/home
- Git: https://gitlab.freedesktop.org/pipewire/pipewire

Firewire

- If you have one, your best bet is http://www.ffado.org/

Notes

- 0040 - When I say sound card, most computers have one build in these days, eg: onboard audio. Physical discrete cards are mostly a thing of the past.
- 0250 - Sound card multiplexing also often called hardware mixing.
- 1240 - There is also a “Pro Audio” mode for sound cards that splits all the channels
- 1705 - Most of these disconnection issues are now fixed as of the time of publishing!

- I'll add more notes as I remember when rewatching this.

- Please note that due to classes and school and coop, the filming/editing/uploads of my videos are very delayed, and might not be the most sensitive. This video was filmed April 2021, Edited June-July 2021, Description written August 2021. I hate writing descriptions and thumbnails…

Contact me

Watch this video on Peertube: https://peertube.tonytascioglu.com More info is probably on my wiki:

https://wiki.tonytascioglu.com

Email: tonytash@pm.me (not monitored 24/7) I might not get to comments on this video until the end of my next school/work term, feel free to post anyways.

I hope you enjoyed the video and learned something!

Shoutouts

Randy MacLeod (and the rest of the Wind River Linux userspace team). I know you had asked me about Pipewire at some point, and I already had this video in the works, so hopefully you find it useful :)

Corrections

- I'll update this as corrections are pointed out.

2021-08-19 17:33 · Tony

HD Radio with RTL-SDR

As someone who sometimes tinkers with radios, I sometimes just scroll through channels with a cheap r820t2 dongle and gqrx.

Intro

Spectrum

The latest thing I noticed was bars of what is presumably digital signals alongside of some commercial FM radio stations.

(todo: add screenshot here)

I questioned whether it was interference, but ruled that out since it wouldn't be in such a clean pattern, and the very consistent bandwidth is very reminiscent of digital radio signals.

Digital Radio

At the time, the only digital radio standard I knew of was DAB, which is used in some parts of Europe.

Turns out, there's an (IMO crappier) standard for digital radio used in North America called HD Radio.

Some downsides I see

Proprietary

Unlike the rather open DAB standard, HD Radio is proprietary, which is enough for me to hate it already.

This means that if you even want to receive or broadcast HDr, you'll probably get hit with hefty fines.

It's also going to (IMO) slow the rate of adoption, since if you're going to spend so much to upgrade a station, who is going to listen to it?

Needs new $$$ receivers

The reason AM/FM radio still works and is kind-of used in the 21st century is that it's ridiculously simple to receive (more so for AM then FM), and for cheap.

Digital radio already has a major hurdle there. If I'm going to spend $100 on a new radio, that's not going to happen *(unless it's for ham radio).

Practically everyone has an AM/FM receiver in their house, probably in an old CD player, boombox or car. Why would someone spend that much money on a new radio?

At that point, you can just pay for Deezer/Spotify.

I mean, I get why it's more expensive, just like DAB, you need what are basically low power CPUs, since you're decoding a digital compressed audio stream.

AAC vs MP2

Adding to the cost and proprietary nature is that HD Radio uses what basically amounts to an AAC audio stream.

Now, at least HE-AAC isn't terrible at low bandwiths, and can at least usually do closer to fullband?

DAB for comparison uses MP2. Some users may be familiar with MP2 as some DVDs used it instead of AC3 for the audio track.

As a pro to HDR vs DAB, MP3, like MP3, is a rather old codec. I find it's only usable at ~ >192 kbit/s, but suffers far worse then AAC at low bitrates.

This means that theoretically at least, a 64 kb/s AAC stream is better then MP2.

On the other hand, AAC is very very patent encumbered.

MP2 has the advantage that all of it's corresponding patents have since expired so anyone can use hardware/software for MP2 without nasty royalties.

AAC is expensive to implement and I guess just hope to not get sued if you're using it.

OPUS really is ideal here - royalty free, which means you can get more widespread adoption, and it crushes the other codecs at low bitrates.

Speaking of sound at low bitrates:

Limited stream bandwidth

From what I've seen, it's also limited to ~96 kbit/s transmissions with up to 4 streams.

This means that stations need to allocate that limited digital bandwidth among all the content they want to stream.

I've seen some stations do a main stream at 64 kbit/s and a secondary at 32, and other stations that do 32/32/32. I've also seen 48/16/16.

That's not a lot of bits to go around.

I have yet to see a station use all four and I'm sure it would sound terrible if they did.

As mentioned above, AAC really starts to break down at 64 kbits and below.

HD? More like compression

When I first read the name “HD Radio” I assumed it was some form to do lossless or high bandwidth audio signals.

“HD”. What a joke that was.

True HD radio might be neat, since while FM doesn't have (digital) compression artifacts-, the volume levels are compressed with that is probably a brick-wall compressor–.

-except the few stations where I can clearly tell that they are just broadcasting off a s—y 64 kbit/s mp3 stream…

–ie: but a multiband compressor, and crank the ratio dial to the right.

I mean, I can kinda see why. You need to get a high SNR so people further away can actually hear your stream, but don't want to pass the max modulation limit so the FCC doesn't come knocking.

Digital on the other hand is either all there or none, so you could get much higher sound quality.

Digital TV with MPEG-2 basic compression allowed us to send 720p60 and 1080i60 signals after all, which was a huge step up from NTSC/ATSC*.

*That's also because video can be compressed digitally more then audio generally but the point stands. no random noise in a DTV signal.

Anyways, back to my point. HD-Radio uses up to what I counted to be around 100 kbit/s of audio.

Even with one stream, that would sound bad with mp2 and mp3, usable with vorbis and aac, and pretty good with opus.

But of course, stations like having a substream so most of them use 64 kbit/s or even 32 kbit/s for their main stream.

As advanced as AAC is, it still sounds terrible at 32 kbit/s. Even at 64, it sounds noticeably **worse** then the FM counterpart.

I heard a talk news AM station in a sub-station at 16 kbit/s, I guess it's usable for voice, but still sounds terrible.

For comparison, OPUS is also fine at around 64 kbit/s. You can hear the compression, but it's bearable, but it also starts to get worse around 32 for stereo music.

HD Radio sounds way worse overall then FM though. Bonus points for feeding the volume compressed-af signal to the digital stream too, even though it's unnecessary. Back to the loudness wars I guess.

Bandwidth usage

HD-Radio signals take up a lot of space.

I guess we're lucky here is NA that we don't have many stations in urban areas.

In Europe, most stations only get 100 kHz spacing, and at most 200 kHz.

When I was in Istanbul, they had over 100 stations in the city on all even numbers (100.0, 100.2, so on)

(they also had some questionable stations with questionable modulation on some off numbers interfering with everyone else…)

Meanwhile, in Toronto, forget about 200 kHz, from my room, I can only pick up ~15 stations in total. We have the space here to do a digital stream beside fm stations.

What I'm trying to say is that in Europe, allocating almost 100 kHz on the sides of the station will be impossible, since other stations are there.

Compared to DAB, HDr takes up more bandwidth from what I can tell. You'd probably need to use a different part of the spectrum.

Note: You can also do what Norway did and I guess just kill off FM and switch to DAB. I don't recommend this approach.

See my notes above, namely the $$$ receivers part for why.

Why would I listen to it

If I wanted to hear a crappy 64 kbit/s AAC audio stream. I'd just press the listen online button on their website.

Surprise! Most people already have a computer/phone with an internet connection that can play those streams.

Not to mention, by the time I open a bad sounding HDR stream, I'd just play the playlist I WANT, WITHOUT ADS from Deezer/Spotify.

I only put up with ads on FM since it's free to listen to, and FM radios are basically free most of the time.

I ain't paying 100$ to listen to ads at 64kbit/s where I don't even get to pick the music, and there are only 4 stations.

Even if a digital radio was free, I think it would gain limited adoption in this day and age.

Receiving signals on Linux

There is a program on GitHub that uses some reverse engineering to play the modified AAC streams.

You can find it here: https://github.com/theori-io/nrsc5

There's also a Python GUI should you prefer here: https://github.com/cmnybo/nrsc5-gui

It works on the cheap R820t2 receivers without many issues (except for the garbage antenna not receiving anything useful)

I guess the GUI can also show a traffic map and weather radar, if the stations bother broadcasting that. but again, anyone with a receiver fancy enough to show a traffic map probably has a freaking cellphone with google maps that also acts as navigation.

Conclusion

Maybe I'm acting like an old-person here, but the simplicity (and cheapness) of AM/FM made is so prevalent. The fact that it was so prevalent was what made it good.

Need to reach a city in an emergency? Chances are, 70% of them have at least a crappy radio.

Nobody (that I know of) would buy an HD Radio receiver (or DAB receiver for that matter). By the time you start putting a CPU and everything, that's already a basic computer.

For that money, why listed to compressed radio with ADS when you can just fire up a streaming service.

Thanks for reading. Leave any comments/questions down below.

2023-11-07 16:48

Older entries >>