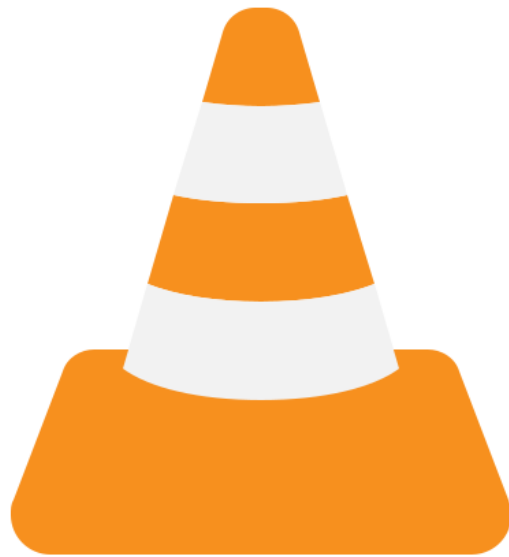




VLC Media Player

A Case Study: Vulnerability History



Lakpriya R.A.K.
IT18000290

Table of Content

Domain and Historical Analysis	2
Product Overview	2
Product Assets	3
Example Attacks	6
Vulnerability History	8
Security Bulletin VLC 3.0.8	8
Design Analysis	10
Architecture Overview	10
Threat Model	16
Assets to Threat Model Tracing	17
Code Analysis	21
Access	21
Demux	21
Access Demux	22
Decoder	22
Video Explaining Link	23

Domain and Historical Analysis

Product Overview

VLC is a cross-platform multimedia player and framework that is free and open source. It can play most multimedia files as well as DVDs, Audio CDs, VCDs, and many streaming protocols, among other things. The VLC multimedia player is a product of VideoLAN project. The VideoLAN project is comprised of a group of volunteers who believe in the ability of open source to revolutionize the multimedia industry. Support for the VideoLAN project has been provided by the VideoLAN organization, which is situated in Paris, France. The VideoLAN non-profit organization is comprised of a committee and members who are all involved with the VideoLAN project in some way. The current board of directors of the VideoLAN organization is comprised of the following individuals: President: Jean-Baptiste Kempf, Vice President: Konstantin Pavlov, Treasurer: Denis Charmet, Secretary: Felix Paul Kühne. VideoLAN develops free software for multimedia that is distributed under the terms of the GNU General Public License. Contributions to the project are welcomed by VideoLAN. Users can contribute time (development, documentation, packaging, tests, user support, ...), material or even money.

In 1996, the project began as a student initiative at the French École Centrale Paris, where it continues today. École Centrale Paris agreed to make it Open Source in 2001, after a thorough overhaul in 1998. This was made possible by an agreement with the École Centrale Paris. The project began to open up to developers outside of the École when it was first launched. It has now grown into a global initiative with developers from more than 40 nations. Because the initiative has been totally isolated from the École Centrale Paris since 2009, it is now managed by an independent non-profit organization.

The product of VideoLAN, VLC source code is freely available in GitHub. This repository consists of over 90,000 contributors and over 500 contributors. Official GitHub page does not accept pull requests but there is a procedure to send patches for this open-source product. The VLC media player mainly written in C language with the help of other languages like C++, Objective C, QML, Makefile, Lua, etc. VLC is distributed under the GNU General Public License version 2 (or later). Because of the licenses of the dependencies, it is de facto GPLv3 on various platforms, such as Linux. In the case of libVLC, the engine is licensed under the

LGPLv2 license (or later). This enables the engine to be embedded in third-party applications while yet allowing such applications to be licensed under different licenses.

This open-source product consists of vast number of documentations for users, developers, and also for hackers too. The main documentations are VLC user guide, VLC documentation for mobile devices, and VLC streaming documentation. In the hacker's guide we can find documentations for Audio Filters, Core, Module source tree, and VLM internals. There are also number of forums for discussing about VLC. This software has downloads over 85,000,000 and supports all major platforms including Windows, Linux, IOS, and android.

Product Assets

Since this is an open-source product most of the source code and related details are publicly available. This product is not a web-based product, and it has to be installed on the device which is being used. Because of that, many of attackers try to extract assets from users by exploiting the VLC software. Since this is a media player users use it to watch videos and play music files. Some of these files can be unreleased testing files. If those files are compromised by an attacker, the market value of the original product can be degraded. If the user has created some playlisted according to their preferences leakage of those kind of information also can be taken as violation of the privacy of the user.

When it comes to VLC software the most important part is core which is located under the “src” file in the repository. libVLCcore is the name of the core of the VLC media player. It is responsible for managing the threads, the modules (codecs, demuxers, and so on), the layers of the modules, the clocks, the playlist, and any other low-level control in VLC. Among its responsibilities is the synchronization management of all audios, video, and subtitle tracks for a given project. Over the top of libVLCcore, there is libVLC, which provides access to all of the core's functionality to third-party application developers. Modules must be linked with libvlccore in order to communicate with the core.

VLC employs **modules** to perform the majority of the work at each stage of the pipeline. Modules are loaded in the appropriate order during runtime based on their requirements. Every module has a unique set of characteristics that are tailored to a specific use-case or a specific environment in which it is deployed. Furthermore, the majority of VLC's portability is achieved by the development of “audio_output/video_output/interface” modules that are specific to the

platform. Functions in “src/modules/modules.c” are responsible for loading and unloading **plugin** modules on the fly. Modules can also be incorporated directly into the application that makes use of libVLC, for example, when VLC is running on an operating system that does not support dynamically loadable code (for example, when VLC is running on Windows). They are referred to as **builtins** in this case. When working with **source code**, modules are typically found in the “modules/ subdirectory”.

VLC uses complex multi-layer input module. The goal behind the input module is to handle packets as though they were known to contain something valuable. It does nothing more than receive a packet, read its ID, and deliver it to the decoder at the precise moment provided in the packet header (SCR and PCR fields in MPEG). All of the fundamental browsing actions are carried out without ever having a look at the content of the elementary stream. Every time a file is read, a new input thread is created. Indeed, input structures and decoders may need to be re-initialized due to the possibility that the stream's specificities will be changed. When the interface thread needs to create a thread, it calls “input_CreateThread” (playlist module). By invoking the function “module->pf_activate”, vlc searches for an access or “access_demux” module to open the stream, which is then used to play the stream. The module thread is initiated, and vlc can begin searching for a demux module to demultiplex the output of the access module if it is successful in its attempt.

VLC is a heavily multi-threaded program. Due to the fact that decoder preemptibility and scheduling would then be a significant problem, the single-threaded technique would have created too much complexity (for instance, decoders and outputs must be kept separate; otherwise, it cannot be guaranteed that the frame will be played at the correct time of presentation.). A multi-process method was also ruled out due to the fact that multi-process decoders typically incur greater overhead (because to issues with shared memory) and that communication between processes is more difficult. The threading structure of VLC is based on the POSIX threading paradigm (pthread). However, for portability considerations, VLC does not directly use the pthread_* methods, but rather a bespoke set of APIs that are comparable to those of pthread_*.

Table 1: Source tree of VLC

Directory Name	Directory Explanation
bindings	Java, CIL and Python bindings
contrib	For required libraries (contains Makefiles to automatically download and compile (or cross-compile) and patch those).
doc	Documentation
include	Header files for VLC
lib	Contains all LibVLC control code
m4	Macro files needed for automake and autoconf
modules	The most important directory besides src/.
po	i18n (language translation) files
projects	Projects based on libvlc, Mozilla plugin, ActiveX plugin and Mac OS X Framework
share	Icons, scripts to make VLC the default player etc.
src	The most important directory besides modules/.
test	scripts to see if everything is OK
extras/analyser	Contains some code style editor (vim, emacs) macro and some valgrind suppressions
extras/buildsystem	Contains alternative buildsystems
extras/deprecated	Contains deprecated files
extras/misc	Contains files that don't fit into any other category
Extras/package	Contains distribution specific files such as ipkg, different rpm spec files, win32 and Mac OS X installation files.

Example Attacks

As mentioned before since VLC media player is a software which has to install on the computer attackers have various types of attack vectors than exploiting web application. Most of the times these kinds of software are exploited by remote code execution like attack mechanisms. VideoLAN has their own GitLab repository for report issues and suggestion of patches. For up to now team has closed more than 18,000 issues and remaining are about 3,000 issues. Most of the issues are related to media playback but there are serious issues of hackers try to break into the software using different video playback codecs and using subtitle formats. Most of the time attackers try to execute codes inside the executable and gain access to the victim's PC to retrieve user data and important files and folders.

A potentially major security weakness has been uncovered in the media player's PC version in 2019, which opens the door for hackers to use the player to execute malicious code on the system. According to reports, the bug in VLC can be used to conduct a denial-of-service attack, corrupt files, steal data, and do a variety of other malicious actions. However, there have been no instances of the bug being exploited to date, and a fix is now being developed to address the issue.

According to CERT-Bund, the vulnerability was discovered in VLC version 3.0.7.1, and it presently has a NIST threat score of 9.8 out of 10, indicating that it is considered critical. The latest VLC security weakness, which has been assigned the CVE-2019-13615 designation in the National Vulnerability Database, can be exploited by tricking users into playing a malicious MKV video file. As a result, while some publications advise users to uninstall VLC until the patch is released, it is likely safe to simply avoid playing any MKV format files that are not trusted.

According to a report by the register, a proof-of-concept film that exploits the vulnerability causes the VLC media player to malfunction. Developer comments on the official VideoLAN bug tracking forum, on the other hand, claim that the VLC crash result cannot be reproduced in large numbers and that it is only functional when the 'Loop One' feature is activated on the VLC Windows version.

For those concerned about potential dangers, the issue can be exploited by a malicious party to remotely execute dangerous code and cause damage ranging from data theft to service

disruptions. According to our records, there have been no complaints of the VLC security weakness being exploited maliciously. Another thing to keep in mind is that the vulnerability affects just the Windows, UNIX, and Linux versions of VLC, not the VLC client for Mac OS X or other platforms. VideoLAN expressed dissatisfaction in a tweet with the fact that it was not notified before the vulnerability was publicly disclosed by vulnerability trackers.

VideoLAN has acknowledged the problem (Issue number #22474) and sent a patch, which is successful in mitigating the flow. The firm that created the VLC media player, interestingly enough, has denied that the flaw can be reproduced in order to cause the media player to crash at all, and the same message has been repeated by a couple of VLC developers as well.

Recently VideoLAN have acknowledged some of remote code executions on the pc version of VLC player. Those issues have documented under the security section of videolan.org website. In the 2020 year about 5 vulnerabilities have been identified under the remote code executions. These vulnerabilities have been named as “VideoLAN-SB-VLC-309” up to “VideoLAN-SB-VLC-313”.

From these vulnerability series attackers are able to build a specially constructed file that would cause a variety of problems. By using a specifically crafted playlist and deceiving the user into interacting with the pieces of that playlist, it is possible to cause remote code execution. This is addressed in greater detail in the reporter's story, which is available online. It is also possible to cause read or write buffer overflows by using specially constructed files or by conducting a Man-in-the-Middle attack against the automatic updater.

If successful, a malicious third party could cause either a crash of VLC or the execution of arbitrary code with the rights of the target user if the attack is successful. While each of these flaws is most likely to cause the player to crash on its own, we cannot rule out the possibility that they could be used in conjunction to leak user information or remotely execute code. ASLR and DEP both help to limit the likelihood of code execution, although they can be circumvented. Luckily there are not any complaints across any exploits that allow for code execution using this issue. These issues are addressed by the team and have patched successfully.

Vulnerability History

When it comes to VideoLAN organization they have been maintaining their main vulnerability flows from 2007. They have a good naming convention for their own vulnerabilities. The vulnerabilities have been divided to two parts. They are Security Bulletins (SB) and Security Advisories (SA). Security Bulletins are associated with each VLC release and can contain a variety of security issues, both internal and external to the VLC software. The naming convention goes as VideoLAN-SB-VLC-***. Mostly last three digits are based on version that vulnerability found of.

Link to the all the issues that VideoLAN have addressed: <https://code.videolan.org/videolan/vlc/-/issues?scope=all&state=opened&search=buffer>

Here I have addressed some of the security bulletins acknowledged by the VideoLAN:

Security Bulletin VLC 3.0.8

Table 2: VideoLAN-SB-VLC-308

Summary	Multiple vulnerabilities fixed in VLC media player
Date	August 2019
Affected Versions	VLC media player 3.0.7.1 and earlier for most issues
ID	VideoLAN-SB-VLC-308
CVE references	CVE-2019-13602, CVE-2019-13962, CVE-2019-14437, CVE-2019-14438, CVE-2019-14498, CVE-2019-14533, CVE-2019-14534, CVE-2019-14535, CVE-2019-14776, CVE-2019-14777, CVE-2019-14778, CVE-2019-14970

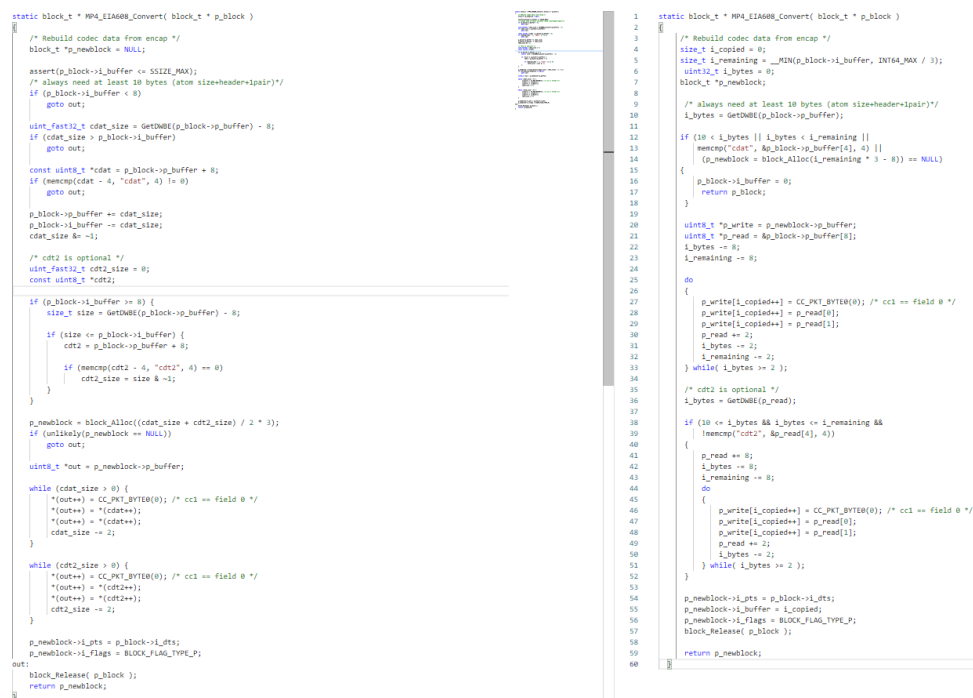
This vulnerability can be used by a remote user to create special files to trigger multiple issues like buffer overflows, and division by zero kind of errors. An integer underflow in the “MP4_EIA608_Convert()” path “modules/demux/mp4/mp4.c” in the version 3.0.7.1 allows attackers to remotely caused a denial of service (heap-based buffer overflow and crash) or possibly have unspecified other impact via a crafted .mp4 file.

Source code commit: <https://code.videolan.org/videolan/vlc/-/issues/11275>

The vulnerability was started from below code snippet:

libsamplerate_plugin!vlc_entry_license__2_1_0a+0x1b17:

As for the mistake the developers have added a function that can divide integers, but they were forgot to check whether the function input validates the input whether it is zero or not. This could easily find by the debugging program and the user who complaint about the program showed it using debugging. Also, this could reach by the validation process in the SDLC. This vulnerability was fixed within 2 months of period. The issue was complaint by two users and two developers of security team managed to fix the issue.



```
static block_t * MP4_EIAG68_Convert( block_t * p_block )
{
    /* Rebuild codec data from encap */
    block_t *p_newblock = NULL;
    assert(p_block->i_buffer <= SIZE_MAX);
    /* always need at least 10 bytes (atom size+header+ipair) */
    if (p_block->i_buffer < 8)
        goto out;

    uint_fast32_t cdat_size = GetInt32(p_block->p_buffer - 8);
    if (cdat_size > p_block->i_buffer)
        goto out;

    const uint8_t *cdat = p_block->p_buffer + 8;
    if (memcmp(cdat - 4, "cdat", 4) != 0)
        goto out;

    p_block->p_buffer += cdat_size;
    p_block->i_buffer += cdat_size;
    cdat_size &= ~1;

    /* cdt2 is optional */
    uint_fast32_t cdt2_size = 0;
    const uint8_t *cdt2;

    if (p_block->i_buffer >= 8) {
        size_t size = GetInt32(p_block->p_buffer - 8);
        if (size <= p_block->i_buffer) {
            if (cdt2 = p_block->p_buffer + 8;
                if (memcmp(cdt2 - 4, "cdt2", 4) == 0)
                    cdt2_size = size & ~1;
        }

        p_newblock = block_Alloc((cdat_size + cdt2_size) / 2 * 3);
        if (unlikely(p_newblock == NULL))
            goto out;

        uint8_t *out = p_newblock->p_buffer;

        while (cdat_size > 0) {
            *(out++) = CC_PKT_BYTE(0); /* ccl == field 0 */
            *(out++) = *(cdat++);
            *(out++) = *(cdat++);
            cdat_size -= 2;
        }

        while (cdt2_size > 0) {
            *(out++) = CC_PKT_BYTE(0); /* ccl == field 0 */
            *(out++) = *(cdt2++);
            *(out++) = *(cdt2++);
            cdt2_size -= 2;
        }

        p_newblock->i_pts = p_block->i_pts;
        p_newblock->i_flags = BLOCK_FLAG_TYPE_P;
    out:
        block_Release( p_block );
        return p_newblock;
    }
}

static block_t * MP4_EIAG68_Convert( block_t * p_block )
{
    /* Rebuild codec data from encap */
    size_t i_copied = 0;
    size_t i_remaining = _MIN(p_block->i_buffer, INT64_MAX / 3);
    uint32_t i_bytes = 0;
    block_t *p_newblock;

    /* always need at least 10 bytes (atom size+header+ipair) */
    i_bytes = GetInt32(p_block->p_buffer);
    i_bytes = GetInt32(p_block->p_buffer);

    if (10 < i_bytes || i_bytes < i_remaining) {
        memcpy("cdat", &p_block->p_buffer[4], 4) ||
        (p_newblock = block_Alloc(i_remaining * 3 - 8)) == NULL)
    {
        p_block->i_buffer = 0;
        return p_block;
    }

    uint8_t *p_write = p_newblock->p_buffer;
    uint8_t *p_read = &p_block->p_buffer[8];
    i_bytes = 0;
    i_remaining = 8;

    do
    {
        p_write[i_copied++] = CC_PKT_BYTE(0); /* ccl == field 0 */
        p_write[i_copied++] = p_read[0];
        p_write[i_copied++] = p_read[1];
        p_read += 2;
        i_bytes += 2;
        i_remaining -= 2;
    } while (i_bytes >= 2);

    /* cdt2 is optional */
    i_bytes = GetInt32(p_read);

    if (10 < i_bytes && i_bytes < i_remaining &&
        !memcmp("cdt2", &p_read[4], 4))
    {
        p_read += 8;
        i_bytes += 8;
        i_remaining -= 8;

        do
        {
            p_write[i_copied++] = CC_PKT_BYTE(0); /* ccl == field 0 */
            p_write[i_copied++] = p_read[0];
            p_write[i_copied++] = p_read[1];
            p_read += 2;
            i_bytes += 2;
            i_remaining -= 2;
        } while (i_bytes >= 2);

        p_newblock->i_pts = p_block->i_pts;
        p_newblock->i_buffer = i_copied;
        p_newblock->i_flags = BLOCK_FLAG_TYPE_P;
        block_Release( p_block );
        return p_newblock;
    }
}
```

Figure 1: Vulnerable code change

In the figure 1, left hand side code is the one vulnerable to divide by zero attack. In the right-side code, they have completely re-written the code. Hence, we could not do a comparison what they have changed we can definitely say this company more serious about their security. When it comes to this attack all of core security principles have been compromised. Since after exploiting the vulnerability attacker can run various code executions and gather information of the victim. This will violate the confidentiality. Secondly, attacker can change files. This means violation of

integrity of the system and lastly, user can commence attacks from remotely and system can be compromised. This means compromising the last core principle which is availability.

Design Analysis

Architecture Overview

The core engine of the VLC media player is “libVLC”. This also works as an interface to the multimedia framework that VLC media player is based on. Core function is working by loading plugins into libVLC at runtime because the library has been modularized into hundreds of plugins. This architecture provides developers with a tremendous deal of flexibility (both VLC devs and devs consuming the library). It enables developers to create a diverse range of multimedia apps that take advantage of the VLC features. libVLC is a C library that can be included in a wide range of applications. It is compatible with the majority of popular operating systems, on both mobile and desktop devices. Under the LGPL2.1 license, it is available for use. The versioning of libVLC is inextricably linked to the versioning of the VLC application. Version 3 of libVLC is the most recent stable major version, and version 4 is the most recent preview/development version. Several programming language bindings for libVLC are available, allowing users to integrate the library into their preferred ecosystem with ease.

In figure two we can identify the file architecture of the VLC player. Most of the files are basic files which come to any project. We can see the modules which are used to build the core function in the system. When talking about subsystems of the VLC player it is mostly referred to these modules. Other than that, we cannot identify any kind of important files in the system.

When further dive into the modules folder we can identify all the building blocks of the core function. Previously discussed vulnerability also resided in one of these modules. When considering these modules, most of the modules are for performing video rendering purposes. Among these modules we can observe there are files like keystore, control, and logger which are used for security functions of the software. Obviously keystore module is used to store keys which are being used in the system and logger is used to make logs of the system like syslogs. Controller module is used to control some of the module inputs to the core which is not much related to security but compromising this kind of a module allows attacker to commence much more damage to the system.

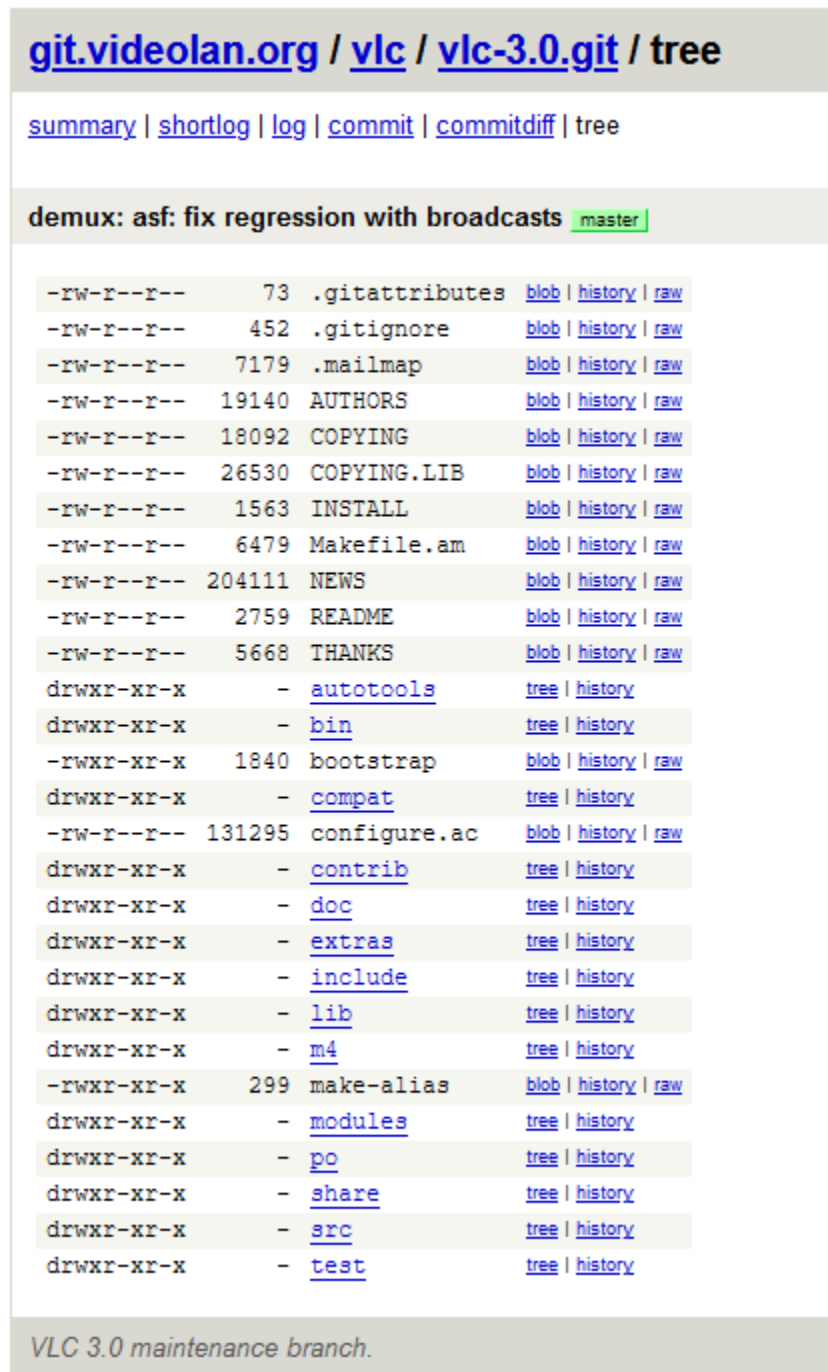


Figure 2: VLC tree

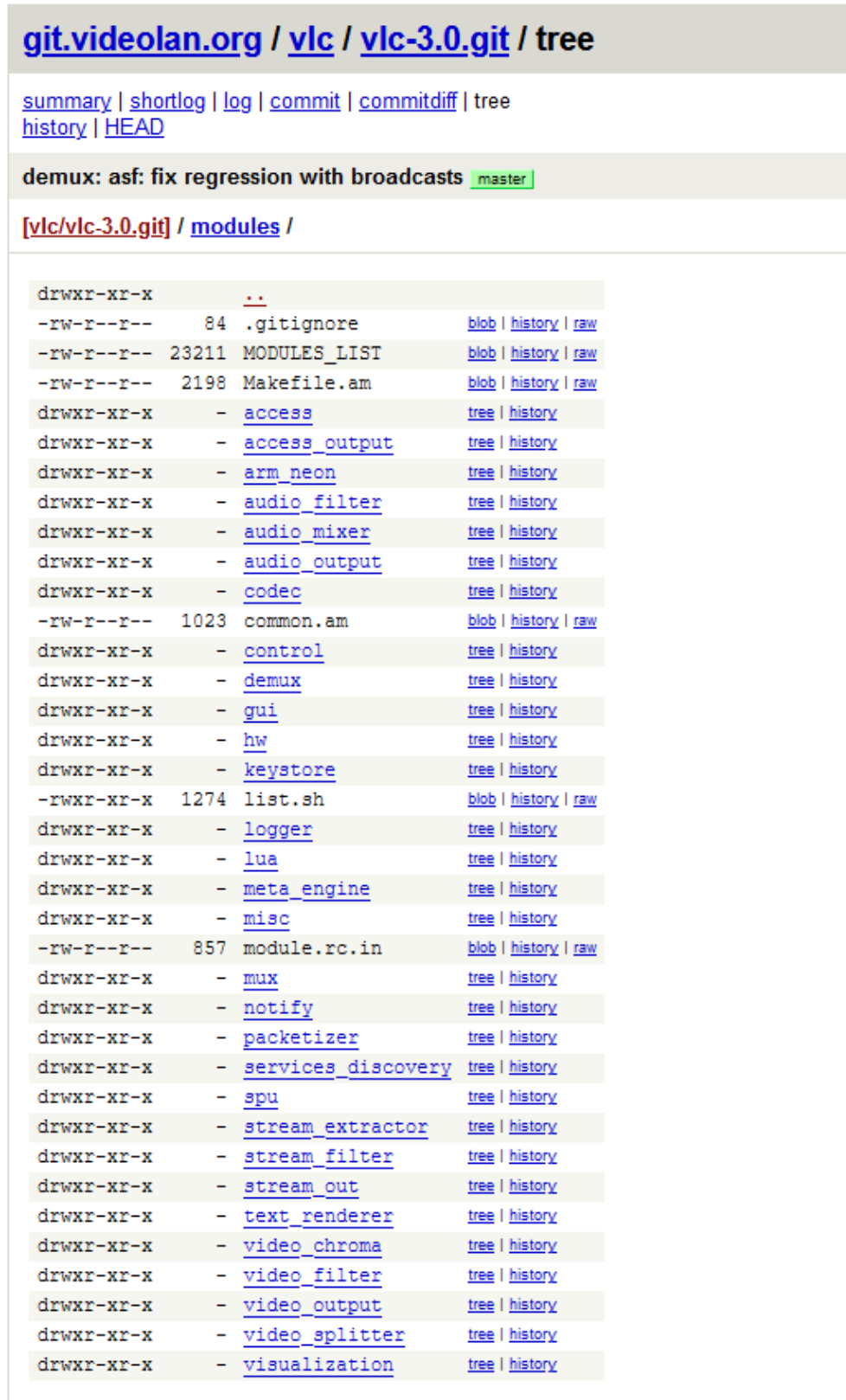


Figure 3: Module tree

From the keystore module we can find out about 11 submodules which are optimized to store keys and perform encryption for necessary files. All of these modules are written in c language. Using a keystore system, can store cryptographic keys in a container, making it more difficult to extract them from the device later on. Once keys have been stored in the keystore, they can be utilized for cryptographic operations without the key material being made available for export. Also included are features that allow to restrict when and how keys can be used, such as requiring user authentication before using a key or allowing keys to be used only in specific cryptographic modes, among other things.

When considering the logger module, we can find out what are the log types of this software uses. These logs are more important when it comes to error reporting and other tasks. Syslog is a protocol for transmitting and receiving notification messages from a variety of network devices in a specific format, known as the Syslog message format. Time stamps, event messages, severity, host IP addresses, diagnostics, and other information are included in the messages. In terms of its built-in severity levels, it can communicate at a variety of levels ranging from level 0, which is an Emergency, through level 5, which is a Warning, System Unstable, critical, and levels 6 and 7, which are Informational and Debugging, respectively. Furthermore, Syslog is an open-ended protocol. Syslog was created to monitor network devices and systems in order to send out notification messages if there are any issues with their operation—it also sends out alerts for pre-notified events and monitors suspicious activity via the change log/event log of participating network devices. Syslog is a network monitoring system that monitors network devices and systems in order to send out notification messages if there are any issues with their operation.

In addition to syslog there are two additional log files which are console.c and journal.c. These files also logging data which are necessary to error handling and reporting. Again, all these files are written using c language. Both mentioned console and journal files are further extension of the syslog file. Both have imported the syslog headers. Journal files are very important when a system crash or system shutdown unexpectedly. All the changes that have not been saved will be logged in the journal module and program can recover more quickly without corrupting.

Console log is used to show alerts in the event of system faults or other situations that necessitate intervention. Figure four will display VLC uses a color scheme to display error messages in their software. In the figure five we can see how they are use colors for error reporting.

```

37 #ifndef _WIN32
38 # define COL(x,y) "\033[" #x ";" #y "m"
39 # define RED      COL(31,1)
40 # define GREEN    COL(32,1)
41 # define YELLOW   COL(0,33)
42 # define WHITE    COL(0,1)
43 # define GRAY     "\033[0m"
44 static const char msg_color[4][8] = { WHITE, RED, YELLOW, GRAY };
45

```

Figure 4: color scheme for console

```

55 flockfile(stream);
56 fprintf(stream, "["GREEN"%0*PRIxPTR GRAY"] ", ptr_width,
57         meta->i_object_id);
58 if (meta->psz_header != NULL)
59     fprintf(stream, "[%s] ", meta->psz_header);
60 fprintf(stream, "%s %s%s: %s", meta->psz_module, meta->psz_object_type,
61         msg_type[type], msg_color[type]);
62 vfprintf(stream, format, ap);
63 fputs(GRAY"\n", stream);
64 funlockfile(stream);
65 }
66 #endif /* !_WIN32 */
67
68 static void LogConsoleGray(void *opaque, int type, const vlc_log_t *meta,
69                             const char *format, va_list ap)

```

Figure 5: Use of color scheme

When it comes to modules most of these modules are dependent on another modules. Since we can see they have imported one module header into other module code. Because of this reason if there is one security weak module is there it can be a reason to compromise the whole system. The vulnerability describe in the vulnerability section is also kind of an example to this point. There was a coding error in one module, and it was resulted to attackers to execute remote codes and compromise the whole system.

Since there are hundreds of small modules in the system and those systems are developed by various developers these modules are more error prone in code level. In the figure six we can see the technical diagram of the VLC. The base of the system is modules. Now we can understand what happens if a module can be compromised since, compromising of one module can be result in compromising other modules as well.

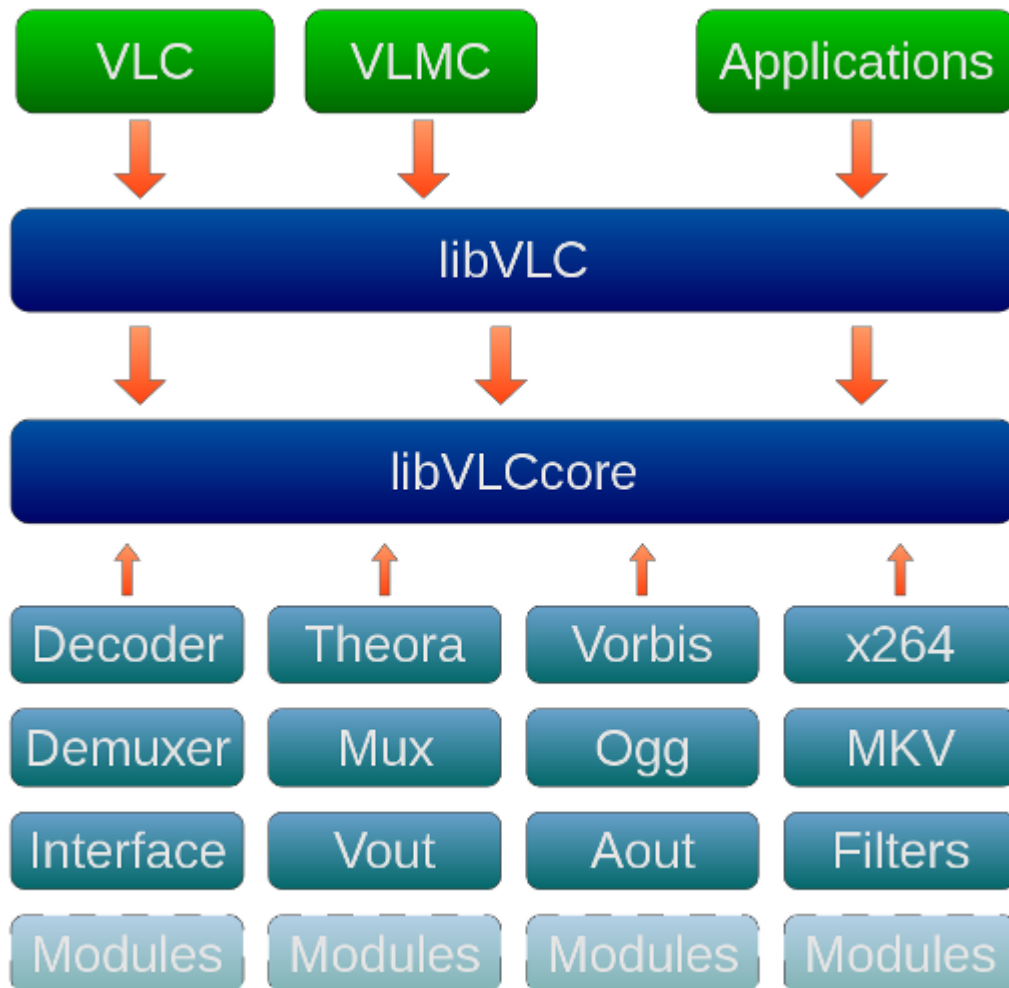


Figure 6: Technical diagram of VLC

In this software we can say very poor security system built into their architecture. We cannot identify dedicated security components in this system. Most of the times modules can use of other modules with elevated permissions. Since this factor most of the time this software vulnerable to attacks like buffer overflow, invalid pointer attacks, specially crafted file attacks and much more. Since compromising one module allows to compromising other modules as a chain reaction this will results in compromising integrity of the system. After compromising the software, the reliability of the software also can be compromised. When the attacker compromised the software, they can use remote code execution to compromise the whole system. This will result in compromising whole set of core security functions in the system.

The media and the programming are the two most important objects in the VLC architectural model. On the surface, media represents anything that can be streamed, but programming defines the manner in which the material should be broadcast, among other things. The daemon is made out of two threads. a thread that monitors the condition of the servers (whether they are up or down), a thread that monitors the state of the programming (whether programs are actually streamed or not) and a request that orders be provided again if something goes wrong. Through out the years VLC team have mainly tried to improve their streaming capabilities without much considering about improving the architecture of the software.

Threat Model

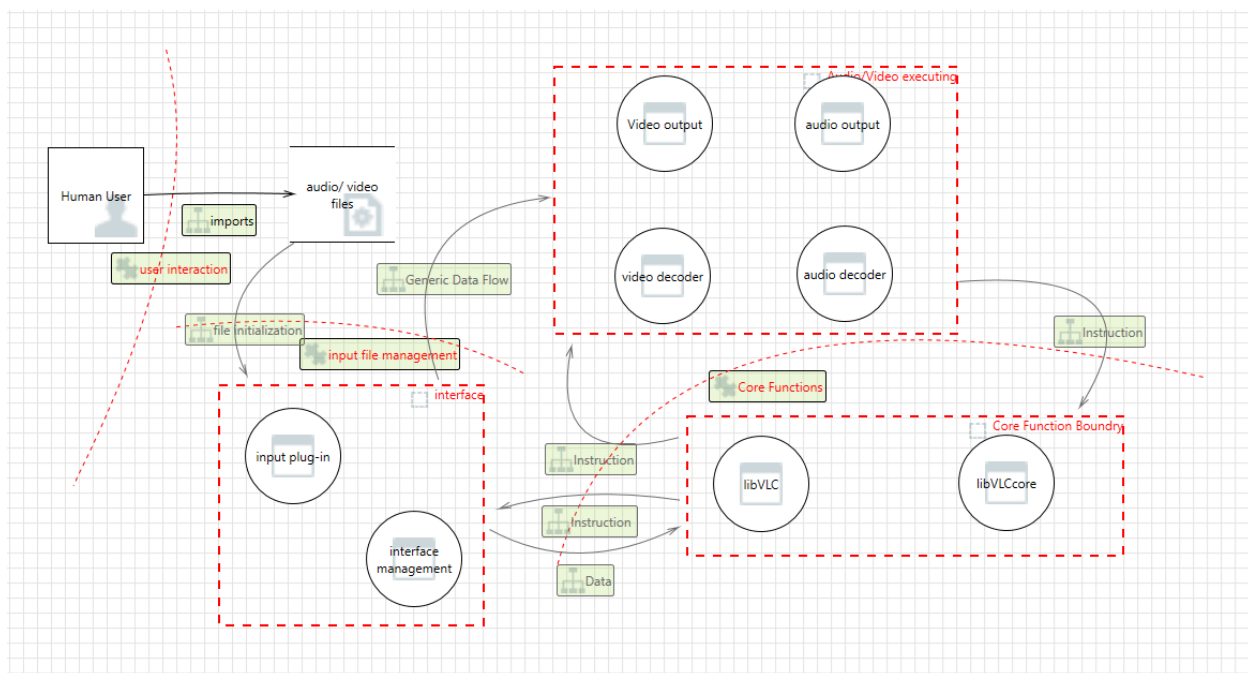


Figure 7: DFD for threat modeling

Here in the figure 7, we can understand the data flow diagram (DFD) for the VLC media player. This is a very basic data flow diagram for the media player which involves human interaction area, basic interface management, video execution, and core function part. But actual player consists of many more modules. This diagram was designed using Microsoft threat modeling tool. In this DFD we can see basically 3 machine boundaries. First boundary is interface boundary. In this part we can identify threads like input plug-ins, output plug-ins, and interface management. These are basically handling the file system and input output files in the system. Next, we can identify the audio/video executing boundary. This part is handling the decoding

functions of the program and giving desired output according to the inputted data file. Finally, the most important process libVLC and libVLCcore are in the core function boundary. This core function boundary restricts the modules to access the core functions of the software. Threads in the interface and other processes are communicating using inter process communication (IPC). The interface boundary is restricted by accessing the processed file in the software in order to protect the plug ins and interface management.

The user interactions also restricted using user interaction boundary to protect the manipulation of inter processes. VLC is comprised of a core and a large number of modules. VLC cannot perform much without modules, because modules are responsible for the majority of the functionalities, we anticipate from this software. As for the example here is how modules are loading. It will scan the default plugins folders that should include VLC modules the first time it is started, then it will create a cache (which will be referred to as the plugins cache) so that the modules may be loaded more rapidly the next time VLC is used. It is possible to group modules together into subdirectories (with a maximum of five levels of organization) beneath the plugins directory. A specific naming convention must be followed by the modules in order for them to be loaded into the VLC media player in recent versions. To avoid confusion, module names should be written in the following format: libmodule_name_plugin.ext where module_name should be the name of your module in lower case and ext should be the shared library extension for the system. Libaccess_http_plugin.dll, for example, is the filename for the “http” access module on a Windows-based computer.

Assets to Threat Model Tracing

This libVLCcore is the name of the program that runs at the heart of the VLC media player. It is responsible for managing the threads, the modules (codecs, demuxers, and so on), the layers of the modules, the clocks, the playlist, and any other low-level control in VLC. Among its responsibilities is the synchronization management of all audios, video, and subtitle tracks for a given project. Over the top of libVLCcore, there is libVLC, which provides access to all of the core's functionality to third-party application developers. Modules must be linked with libvlccore in order to communicate with the core.

As a result, VLC employs modules to perform the majority of the work at each stage of the pipeline. Modules are loaded in the appropriate order during runtime based on their

requirements. Every module has a unique set of characteristics that are tailored to a specific use-case or a specific environment in which it is deployed. Furthermore, the majority of VLC's portability is achieved by the development of `audio_output/video_output/interface` modules that are specific to the platform. Modules for plugins are loaded and unloaded dynamically by functions in the `src/modules/modules.c` directory of the project. Modules can also be incorporated directly into the application that makes use of `libVLC`, for example, when VLC is running on an operating system that does not support dynamically loadable code (for example, when VLC is running on Windows). They are referred to as builtins in this case. When working with source code, modules are typically found in the `modules/` folder.

VLC is a heavily multi-threaded application. Due to the fact that decoder preemptibility and scheduling would then be a significant problem, the single-threaded technique would have created too much complexity (for instance decoders and outputs have to be separated, otherwise it cannot be warranted that a frame will be played at the exact presentation time). A multi-process method was also ruled out due to the fact that multi-process decoders typically incur greater overhead (because to issues with shared memory) and that communication between processes is more difficult. The threading structure of VLC is based on the POSIX threading paradigm (`pthread`). However, for portability considerations, VLC does not directly use the `pthread_*` methods, but rather a bespoke set of APIs that are comparable to those of `pthread_*`.

VLC's asynchronous decoding and playing is another important feature: decoding is handled by a decoder thread and playing is handled by the `audio_output` or `video_output` threads, respectively, in the program. When designing a decoder, it is important to make sure that an audio or video frame is played at the precise proper time and that none of the decoder threads are blocked. As a result, the communication structure between the interface, the input, the decoders, and the outputs becomes increasingly complex.

It is permissible to have several input and `video_output` threads reading multiple files at the same time, despite the fact that the existing interface does not provide a mechanism to do so (this is subject to change in the near future). Whatever the case, the client has been designed with this in mind from the beginning. This also indicates that a non-reentrant library (such as, for example, `liba52` See talk page) cannot be used without first obtaining a global lock.

Presentation All generated samples are dated based on the time stamps located in the system layer of the stream, which is passed to the decoders. The output layers are responsible for ensuring that they are played at the appropriate moment. Dates are translated to microseconds, and an absolute date is the number of microseconds that have passed since the beginning of time (Jan 1st, 1970). The `mtime_t` type is a signed 64-bit integer with the sign bit set.

As for the threat model mapping references and code analysis please follow the below table.

Table 3: Threads in interface management

Threads	
<code>vlc_clone()</code>	This will create a thread
<code>vlc_join()</code>	Waits for a thread to finish and then releases the resources it is using.

Table 4: Mutexes in interface management

Mutual exclusion	
<code>vlc_mutex_init()</code>	This function creates a non-recursive mutex.
<code>vlc_mutex_init_recursive()</code>	This function creates a recursive mutex. (discouraged)
<code>vlc_mutex_lock()</code>	This method locks a mutex and waits if necessary.
<code>vlc_mutex_trylock()</code>	If the mutex is not already locked, this function locks it; otherwise, it returns an error.
<code>vlc_mutex_unlock()</code>	Unlocks a mutex.
<code>vlc_mutex_destroy()</code>	Destroys a mutex.

Mentioned functions are managed by the interface management in the interface boundary. And below have described the core functions in the core boundaries. There will be functions like modules/ which are useful when managing modules. This might confuse as redundant, but it is for better compatibility of the core functions with the external modules since the functions are in two different boundaries. These libVLCcore functions are located inside the src folder. In the table 6 functions belong to video and audio executing will be described.

Table 5: Core functions in LibVLCcore

Core functions	
config/	Modules can read and write to configuration from the command line and configuration file. This module loads configuration from the command line and from a configuration file.
control/	The following functions are used to control the behavior of libVLCcore: play/pause, volume management, Fullscreen, log verbosity, and so on.
input/	A decoder receives elementary streams that have been reconstructed by reading them from an input module and passing them to the decoder (s).
interface/	Code for user interaction, such as keystrokes and device ejection, is contained within this section.
modules/	Used for module management.
network/	Handling the network interfaces, management of sockets, network error handling.
osd/	Manipulation of the display on the screen

Table 6: Audio/video execution

Audio/video	
audio_output/	When the audio mixer is initialized, i.e. when it determines the appropriate playing frequency, it resamples audio frames received from the decoder.
video_output/	When the video display is first turned on, it loads all of the pictures and subpictures (such as subtitles) from the decoder(s), and then either shows them as is or converts them to another format (such as YUV to RGB).
playlist/	Playlist interaction, such as stop, play, next, and random playback, is managed by this class.
stream_output/	It has the ability to stream audio and video across the network.

Code Analysis

In this part has described four main modules that are frequently used by the VLC media player which are Access, Demuxer, Access Demuxer, and Decoder.

Access

Access capability modules are intended to be the first and last items in a module chain, with the first and last elements being identical. When an MRL is supplied into them, they output a bitstream that can be sent into a Demuxer for further processing. The majority of VLC's fundamental I/O is handled by the access input and output functions. Most of the time, they are protocol implementations (http, ftp,...) or devices that access the internet (Webcams, Capture cards).

Writing access module:

```
set_capability( "access", 60 )
set_category( CAT_INPUT )
set_subcategory( SUBCAT_INPUT_ACCESS )
```

Demux

Different "file" formats, like as AVI and MKV, are handled by the 'demux' modules, which are specifically intended for this purpose. They are often placed after the Access module in the module chain, and they are responsible for creating the various tracks that are transmitted to the decoders. The demuxer modules are responsible for processing the input stream that is sent to them by the core input. It divides the input stream into the several tracks that are needed (called elementary streams). It is technically correct to say that the demuxers are pulling information from the stream, rather than that information is being pushed via access to the demux. WMV/ASF, Ogg, and Mkv are examples of demultiplexers.

Writing demux module:

```
set_capability( "demux", 60 )
set_category( CAT_INPUT )
set_subcategory( SUBCAT_INPUT_DEMUX )
```

Access Demux

A number of modules with the 'access demux' capability are designed to handle both the access stage and a portion of the demux stage in a single unit. Consequently, you must first read the access and demux pages before proceeding forward. An access-demuxer can be thought of as a demuxer that also handles the access at the same time as it performs the demuxing. It is possible that the stream s is null in an access-demuxer. DVD, Bluray, and v4l2 modules are all examples of access-demuxers.

Writing access demux module:

```
set_capability( "access_demux", 60 )
set_category( CAT_INPUT )
set_subcategory( SUBCAT_INPUT_ACCESS )
```

Decoder

The decoder is responsible for the mathematical portion of the stream-playing process. It is separated from the demultiplexers (located in the input module), which manage packets in order to reconstruct a continuous elementary stream, and from the output thread, which takes samples reconstructed by the decoder and plays them back on the computer. A decoder, in its most basic form, has no interaction with other devices; it is entirely algorithmic.

The input thread causes the required decoder modules to be spawned from the src/input/decoder directory. In the CreateDecoder(input thread t *p_input, es_format t *fmt, int i_object_type) function, a_p dec variable of type i_object_type (in our case, VLC_OBJECT_DECODER) is created, and the module Need(vlc_object_t *p_this, const char *psz_capability, const char * module Need creates a list of all of the possible modules for this capability based on a score assigned to each module. As an example, in the a52 module, we can read:

```
set_capability( "decoder", 100 );
```

When we request a decoder, its score is 100, indicating that it is a successful request. The same technique is used throughout VLC to determine which module will be utilized in each section.

After that, it checks to see if the decoder requires packetized data (in that case, it finds and loads a packetizer module). At the end of the program, it calls the following function to start the decoder thread:

```
vlc_thread_create( p_dec, "decoder", DecoderThread, i_priority, false )
```

Video Explaining Link

https://mysliit-my.sharepoint.com/:f:/g/personal/it18000290_my_sliit_lk/EuVqpbqyj9KpsezG-zcTW0BX5TQDkBj8lNv5Kla7pnRUQ?e=iPQSYt