A Case Study: VLC Media Player Vulnerability History

Helena Lynd

Coray Bennett

Hunter White

Abiel Yemane

Table Of Contents

Product Overview:

VLC Media Player (VideoLAN Client) is a free, open-source multimedia player developed by the VideoLAN project. It is renowned for its versatility and capability to play virtually all multimedia file types, including audio, video, and streaming protocols. VLC is platform-independent, with support for Windows, macOS, Linux, Android, iOS, and other operating systems. It uses its internal codecs to decode media, eliminating the need for external codec packs. This feature makes VLC highly reliable and widely used by individuals and organizations.

VLC also supports advanced functionalities such as media conversion, streaming, and even playback of incomplete or damaged files. VLC includes functionality to act as both a client, playing media files from a network, and as a server, enabling it to stream the media files to other devices on the network. This makes it applicable as a key component for client-server applications, including environments like classrooms, businesses, and remote settings where collaboration is needed. Its lightweight design and robust feature set make it a favorite among both casual and professional users.

Users and Purpose:

VLC has the capability to serve a wide variety of users, including:

Casual Consumers

Individuals who seek a simple yet versatile media player. Its capability with most audio and video formats ensures that less technical users do not need to rely on a file converter in order to stream multiple file types.

IT Professionals and Developers

Individuals who will appreciate VLC's advanced capabilities, including the support for a wide range of network protocols, dynamic streaming functionalities, and the ability for command-line interaction/automation. This ability to bypass the GUI allows users to create and use scripts to get the most out of the media player

Organizations

Organizations can utilize VLC for professional purposes such as remote collaborations, training sessions, and other multimedia purposes like presentations. Its reliability, performance, and ability to handle multiple file types and network configurations make it appealing to organizations looking to get the most out of their media player.

The application primarily aims to offer seamless playback of various media formats while serving as a platform for media-related experiments and integrations. As an open-source project, VLC aligns with the VideoLAN project's mission to make media playback accessible to everyone, ensuring global adoption without financial barriers. Its goals extend beyond accessibility, aiming for versatility through constant updates, user empowerment via advanced tools, and maintaining user trust by prioritizing security.

Development & Business Objectives:

The development of VLC Media Player is overseen by the nonprofit VideoLAN project, consisting of a global network of volunteers and contributors. The core developers focus on maintaining and enhancing the application's performance and feature set, while security analysts address vulnerabilities and ensure the platform's safety. Community contributors assist with testing, documentation, and localization, while partner organizations provide funding and resources. As an open-source project, VLC thrives on collaboration, with code reviews and transparent discussions shaping its growth. Development follows an iterative process, incorporating user feedback to prioritize features and fixes. Security is a top priority, with regular audits, bug bounty programs, and swift responses to emerging vulnerabilities.

Overview of Findings:

As people in both professional and personal settings become more reliant on remote collaboration, the capabilities of media players and streaming services will continue to be stretched. It is important for companies and projects to protect against exploitations and respond to vulnerabilities as soon as they are discovered. We have found a list of addressed security concerns from the VLC team, some of which will be discussed in depth in the Vulnerability History section of this paper.

Product Assets:

Table 1: Potential Resources and Vulnerabilities in VLC Media Player

Category	Resource/Asset	Exploitation Potential	Discussion
Protected Data	Media files (e.g., `.mp4`, `.mkv`, `.avi`)	Unauthorized access, corruption, or deletion	An attacker could manipulate VLC to alter or delete sensitive user media files by exploiting the path traversal vulnerability
Protected Data	User settings (e.g. .vlcrc file)	Manipulation of user settings	An attacker could manipulate their own user settings file to escalate privileges, or manipulate someone else's file to add an insecure plugin that the attacker can further exploit

Category	Resource/Asset	Exploitation Potential	Discussion
Product Resource	Codecs used by VLC for decoding media/Parsing multiple file types	Arbitrary code execution	An attacker could create a polyglot file that will be decoded incorrectly by VLC, leading to overflows and arbitrary code execution
Product Functionality	Streaming to other devices	Compromise the integrity and privacy of data	An attacker could perform a man-in-the-middle attack, as VLC media player has the ability to stream data to other devices on it's network
Product Functionality	Accepting streams from other devices	Compromise the availability of VLC and the integrity of user data	An attacker could encode an exploit in a data stream, then remotely send it to the device accepting streams through VLC media player
Product Functionality	CLI Interaction/ Scripting	CLI commands on victims system	An attacker could create a malicious script that executes unwanted commands on the user's system
Product Functionality	Plugins	Larger attack surface	An attacker could exploit vulnerable plugins to execute attacks
Intangible Property	Cross-platform availability	Larger attack surface	Distributing VLC on many different platforms increases the attack surface
Intangible Property	Open-source code	Attackers have more knowledge to create exploits	Having the codebase for VLC publicly available increases an attackers knowledge about the system and how it works

Category	Resource/Asset	Exploitation Potential	Discussion
Intangible Property	Availability	DoS attacks	It is important for VLC to always be available, as organizations make use of it for both in-person and remote work

VLC media player is still a widely used media service with an open source foundation. Its versatility in being able to interpret a wide range of media and files makes it an appealing target for attackers. VLC's support for network protocols and services for streaming makes it even more susceptible to malicious actors.

VLC Media Player stores user-specific data. Users are given an 'open recent media' option, meaning your playback history is being stored. User settings and preferences are able to be customized, meaning these configurations for streaming are stored. VLC may gain access to tokens and other relevant login/authentication information relevant for accessing private media servers.

VLC must be capable of isolating malicious media and scripts from uploaded files to prevent them from affecting the system. The wide range of supported file types make this a point for potential attention.

Example Attacks:

An attacker gains access to the victims device, then edits the VLC user settings file (.vlcrc) to execute a script upon startup. Then, when the user next opens VLC Media Player, the attacker's script is executed. This script could have many different purposes, such as exposing the victims data or further editing their settings to increase the attack surface so that an attacker can gain even more control of their device.

- **Confidentiality:** Highly compromised, as the user's settings are visible to the attacker.
- **Integrity:** Highly compromised, as the user's settings are altered without their permission.
- **Availability:** Not compromised as a result of this attack.

When VLC is being used as a media server, port forwarding is required. An attacker could attempt to access sensitive files via connections to the server. This could be accomplished with path traversal, or any kind of vulnerability relating to exposed memory. This is an attack on one of the key assets on our system, user/organization-specific data.

- **Confidentiality:** Highly compromised, sensitive user information can be exposed.
- **Integrity:** Potentially compromised, if the attacker gains write access and any files are altered as a result of the attack.
- **Availability:** Not compromised as a result of this attack.

An attacker has crafted media files that have been constructed for multiple malicious purposes. These packets are trying to take advantage of the potential improper packet processing with the goal of triggering a buffer overflow. When this malformed media is uploaded, it triggers this overflow during the memory allocation. This corrupted memory creates an environment prone to denial of service or code execution attacks.

- **Confidentiality:** Potentially compromised. If remote code execution is exploited, sensitive files or credentials of the user may be exposed to the attacker.
- **Integrity:** Potentially compromised. This malicious media may modify files or corrupt memory, meaning systems reliability is now questioned
- **Availability:** High level of compromise. The improper allocation can lead to crashes, posing a major threat users ability to access and use the system

An attacker crafts a malicious media file that exploits a vulnerability in VLC's media parsing engine. When the victim opens the file in VLC, the exploit triggers, allowing the attacker to execute arbitrary code on the victim's device. This code could be used to steal sensitive information or install backdoors for future access.

- **Confidentiality**: Highly compromised, as the attacker can access private data on the victim's device.
- **Integrity**: Highly compromised, as the attacker can execute arbitrary code, potentially altering files or installing malware.
- **Availability**: Partially compromised, as VLC may crash during exploitation, interrupting media playback.

An attacker gains access to the victim's device and places a malicious VLC plugin in the VLC plugins directory. The plugin is designed to capture sensitive data or inject malicious code when VLC is opened. When the user next launches VLC Media Player, the malicious plugin executes alongside legitimate VLC functionality. This can be used to log keystrokes, capture screenshots, or even install additional malware.

- **Confidentiality**: Highly compromised, as the attacker accesses sensitive data from the victim's device through the malicious plugin.
- **Integrity**: Highly compromised, as the VLC software behavior is modified to execute the attacker’s malicious code.
- **Availability**: Not compromised, as the attack does not affect VLC’s ability to function normally.

Vulnerability History:

This section details four vulnerabilities that the team found merited deeper discussion.

—---

CVE-2024-46461/VideoLAN-SB-VLC-321 - Buffer Overflow

The latest report for this vulnerability was recorded by Andreas Fobian in June of 2024 on this security bulletin VideoLAN-SB-VLC-321. This security vulnerability was fixed within VLC media player version 3.0.21 release, and relates to a potential overflow vulnerability that could lead to a denial of service attack. This shows that availability is the most immediate and direct category affected by this vulnerability. The mishandling of reallocation during the packet processing not only provides accessibility concerns, but exposes VLC users to potential remote code execution through a malicious media. This issue stems from the insufficient/non-existent boundaries set for checking memory, which is worsened by the lack of proper input handling and media validation. It is noted by the developers that no code execution has been actually observed through this vulnerability through its existence in the system.

CVE-2024-46461 was acknowledged as fixed in the commit on Jan 9, 2024 by Thomas Guillem. A single code review for this fix was merged to master on July 5, 2024. The quality of the code review can be questioned as typos are not recognized and persist throughout the commits and source code (reaced -> reached). This quality also comes into question when you see that 9 commits were merged across 6 developers that directly involve the 10 lines of code that make this vulnerability. The fact that some of these commits include phrases like 'fix a memleak' (commit December 2018) but they just are setting to skip the first 8 bits of p_data, not addressing the memory management. These faults should not have stayed within the system for 7,569 days.

Quality code reviews and merge requests need to be a priority. If misspelling is making it through the cracks, no wonder memory mismanagement was overlooked for these overflow cases. Proper fuzz testing would also ensure the security of the system. By putting the packet processor up against known malicious media, you are able to test these inputs against the system in a safe environment. Even research into better libraries for safe allocation and boundary checking may prove beneficial to the systems safety.

April 20 2003 commit Laurent Aimar

Reallocation without proper checking was introduced on the initial commit of this file. Lines 1292-1296 highlight the absence of proper validation and memory allocation practices. The realloc function is used to resize the memory pointed to p_sys->p_header. If this fails, there is no recheck this pointer, which could lead to memory leaks or potential undefined behavior. There is no validation of the input i_packet_legth. There may exist a state where i_packet_length and i_header are either less than 8 or exceed max size, resulting in an integer underflow or overflow. The blind memcpy of these potentially unverified resources can result in buffer overflow, or remote code execution.

December 28 2015 commit Francois Categnie

This updated code shows promise toward secure memory practices by storing the result of the realloc to a temporary pointer. This prevents any potential losses to the original pointer to memory if the realloc fails. This addresses a security risk of memory leaks. There are still no boundary checks for p_data and i_packet_length so the memcpy of can still corrupt data and lead to unintended consequences. This code still assumes that p_sys->i_header and the i_packet_length - 8 will not exceed the size allowed for this data type.

January 9 2024 commit Thomas Guillem

The addition of the add_overflow method provides proper checks that p_sys->i_header does not result in an overflow. This eliminated the risk of later requesting this incorrect memory allocation, which would lead to the described buffer overflow. This code continues the proper use of the temporary pointer for safe memory reallocation. Although this developer fails to address the typo in p_reached, this still ensures that the original pointer remains intact if the reallocation fails. Thomas then goes on to address the potential unsafe += assignment for p_sys->i_header and now assigns it to new_header_size that is responsive towards incorrect allocations, ensuring that no integer overflow is present.

—---

CVE-2022-41325 / VideoLan-SB-VLC-318 - Stack Buffer Overflow

This vulnerability, reported and fixed by the VLC Media Player team in version 3.0.18, pertains to a stack buffer overflow when processing specific MKV metadata. Exploiting this vulnerability could result in a denial of service or arbitrary code execution. The issue arose from improper bounds checking while processing Matroska (MKV) files. Insufficient input validation allowed attackers to craft a malicious file capable of corrupting the stack, leading to potential code execution or application crashes. The vulnerability was introduced on February 17, 2013, and persisted for approximately 9 years and 7 months until it was fixed on September 19, 2022.

CVE-2022-41325 was reported by 0xMitsurgi from Synacktiv and acknowledged on September 19, 2022 by Romain Vimont.

February 17, 2013, commit Francois Categnie

This commit introduced unsafe assumptions about height, width, and bitsPerPixel values, assuming they would fit within a 16-bit range without implementing adequate assertion checks. Exploiting this vulnerability allowed attackers to repeatedly crash the application, leading to significant user frustration and service disruptions, particularly when VLC was embedded in other systems

September 19, 2022, commit Romain Vimont

The fix replaced the use of **int** with **size_t** for calculations involving width, height, and bitsPerPixel, ensuring these variables could handle a larger, non-negative range of values. This change prevented potential overflows caused by improperly sized types. Additionally, assertion checks were added to validate that these values fell within acceptable bounds before proceeding with calculations, safeguarding the integrity of the processing logic.

These enhancements provided a more robust method for calculating framebuffer size, mitigating risks associated with unexpected or malicious input. By enforcing bounds checking and adopting safer data types, the fix not only resolved the immediate vulnerability but also established a more secure pattern for handling similar computations in the future. This commit exemplified strong defensive programming practices, enhancing the overall security of the VLC application.

To prevent such vulnerabilities, several software engineering practices should be adopted. First, validating all external input ensures it adheres to expected ranges and constraints, minimizing the risk of unexpected behavior. Static code analysis tools can help identify potential buffer overflow vulnerabilities during development. Incorporating assertion checks can safeguard against out-of-bounds variable values. Finally, thorough peer code reviews focusing on edge-case handling and assumptions can detect potential flaws early in the development process.

This vulnerability impacts the core principles of the CIA Triad. It threatens confidentiality by enabling arbitrary code execution, potentially exposing sensitive user data. It compromises integrity by allowing attackers to alter the application's behavior or manipulate its data. Furthermore, it undermines availability by causing application crashes, leading to service disruptions.

—---

CVE-2023-46814 / Security Bulletin VLC 3.0.19

Arbitrary Code Execution via Windows Uninstaller

Both the CVE and the Security Bulletin are quite vague in terms of how this vulnerability works. The impact section of the bulletin states,

"If successful, a malicious third party could trigger an execution of an arbitrary binary on uninstallation of VLC with system priviledges. We have not seen exploits performing code execution through this vulnerability."

The bulletin has some useful info: The vulnerability was fixed in version 3.0.19, and it also specifies that it is an "NSIS uninstaller vulnerability". NSIS is a system used to create Windows installers, which is apparently used for the VLC media player uninstaller for Windows.

Looking at VLC version 3.0.19, we can see the following commit, the most recent commit relating to the NSIS uninstaller:

Commit 236b3184d75a3eeb4a56340b5eeab746c407157f, authored by Steve Lhomme on September 26, 2023:

Commit Title: *nsis: always use the regsvr32.exe from the system*

Commit Message: *We should not use whatever is in the PATH. Especially as the current directory is set the a temporary directory during uninstallation.*

This commit replaces a relative path reference to `regsvr32.exe` with a reference using the `$SYSDIR` path. This provides a lot of clarity as to how the vulnerability may be exploited. Since `regsvr32.exe` was executed using whatever the current directory was, then if the temporary directory that is set during installation is targeted by a malicious party, a different `regsvr32.exe` could be executed on the user's system on uninstall.

The original commit can be found via the git blame:

Commit 35bb5df5bfa263b9865b8b8c9675b39fd08d7f9c, authored by Jean-Baptiste Kempf on April 6, 2008.

Commit Title: *Windows Installer improvements.*

This original commit introducing the vulnerability has several notable points:

1. It was authored by *the* Jean-Baptiste Kempf, who is often attributed as the "creator" of VLC media player. He is one of the top contributors and current president of the VideoLAN organization.
2. This single commit is 830 lines of code.
3. The commit also removes 794 lines of original code, rewriting the entire file.
4. The commit does not include a commit message beyond "Windows Installer Improvements". It does not specify what the improvements are or how they improve the installer.

This vulnerability was present in the VLC media player application for about 15 years since it was introduced, and there were 16 unique authors who contributed to the file between the initial commit and the fix.

In terms of engineering practices, early, smaller and more incremental changes could have helped to prevent this vulnerability. The general practice of avoiding relative paths would have also been a good idea here. It isn't confirmed, but based on the date and lack of a sign-off on the initial commit, it's very likely that the code was never reviewed by another engineer before merge.

An arbitrary code execution vulnerability by its very nature majorly impacts all points of the CIA triangle. Confidentiality is impacted since an arbitrary executable could read sensitive user information. Integrity is impacted for a similar reason, writing to accessible files. Availability can be impacted if a kind of ransom attack is executed and user data is encrypted.

—--

CVE-2021-25803 / VideoLAN-SB-VLC-3013 - Out of Bounds Read

This vulnerability is caused by the improper handling of .avi media files. When these files are processed, there is potential for buffer overflow that can lead to a crash or remote code execution. A maliciously crafted file can be created to take advantage of the vulnerability and always cause buffer overflow.

The specific mistake made from a software engineering perspective was similar to an "off-by-one" mistake. The logic was written to check if the total size of the buffer would have room for the size of the data plus the header, which could pass even when the remaining size in the buffer was not sufficient, causing a buffer overflow. The fix was to instead subtract the size of the header from the remaining buffer size, and change the handling of negative numbers (which only happen in the case of malformed data). This vulnerability could have been found sooner with in-depth code reviews or pair programming, in which another person may have caught the mistake. The vulnerability could also have been found sooner with more unit tests that checked all edge cases, as the exploit involved only a specific edge case of processing certain malformed files.

The vulnerability was originally introduced in this commit on January 12, 2010:

> Commit Title: *Implemented support for embedded subtitles in AVI*
>
> This commit introduced 126 new lines of code, including the vulnerable line that allowed for buffer overflow.

The vulnerable line was not changed until it was fixed in this commit on November 9, 2020:

> Commit Title: *AVI: Fix Integer Overflow*
>
> Commit Message: *Which would in turn cause a size verification failure, leading to a buffer overflow. Reported by: Zhen Zhou, NSFOCUS Security Team*
>
> This commit fixed the vulnerable line by correcting the calculation that checked if there was enough space in the buffer for the data, and fixing the handling of negative numbers.

The vulnerability existed in the system for 10 years and 10 months. Only two developers ever worked on the vulnerable line of code, however twelve developers worked on the file containing the vulnerable line throughout the years the vulnerability was present.

This vulnerability compromises confidentiality, as random code execution can reveal system and user information. The vulnerability could compromise integrity, as random code execution could change variables and stored data that would affect the contents of the system. The vulnerability has no impact on availability.

Architecture Overview:

Input Subsystem

The Input Subsystem is responsible for retrieving media from various sources, such as local files, CDs/DVDs, network streams, and webcams. It works with input modules that understand specific protocols (e.g., HTTP, RTSP, FTP) or hardware interfaces. This subsystem handles raw data acquisition and forwards it to the Demuxer for parsing. Errors here, such as improper validation of URLs or file headers, can introduce vulnerabilities like resource exhaustion or path traversal. The input subsystem faces potential vulnerabilities when handling malformed or malicious input files or streams. For example, insufficient validation of file headers or network data could allow attackers to exploit the player by injecting corrupted data that crashes the system or gains unauthorized access to system resources. This subsystem can mitigate some of these vulnerabilities by implementing proper input handling.

Demuxer

The Demuxer Subsystem extracts individual streams (audio, video, and subtitles) from multimedia containers such as MKV, MP4, and AVI. It includes a variety of demuxer modules, each tailored to specific container formats. This subsystem is complex because it must handle diverse and often proprietary specifications. Due to the variety of file types and specifications that the demuxer must support, make it a possible target for attackers to exploit. Errors in handling inputs and undocumented formats can present security risks within the system. Mistakes in boundary checking or parsing logic, as seen in vulnerabilities like CVE-2022-41325, can lead to stack or heap buffer overflows, enabling remote code execution.

Compromising of the demuxer possibly exposes other subsystems to raw streams with malicious data. This subsystem uses good software practice by implementing different demuxer modules for varying formats, but further distrustful decomposition could be employed to prevent these vulnerabilities from cascading. These demuxer concerns are also highlighted in CVE-2013-1954, where a crafted ASF formatted movie could trigger the out-of-bounds error. As new formats present themselves, and others become irrelevant, there is a high chance that this subsystem will continue to change. Precedents must be set to continue ensuring the security of this module and system as a whole. As video and media file complexity increases, the chance of new vulnerabilities increases. To mitigate risks, VLC developers use safer programming practices and static analysis tools for demuxer code.

Decoder Subsystem

The Decoder Subsystem processes compressed streams from the Demuxer and converts them into raw data for playback. It supports a wide range of codecs, including H.264, VP9, AAC, and more. This subsystem is a high-risk area because decoders often deal with complex, poorly documented file formats, which can lead to vulnerabilities such as integer overflows or memory corruption. Third-party codec libraries used here may also introduce external risks.

A mistake within this subsystem can have consequences affecting the reliability, security, or performance of the application. If memory is mishandled within the decoder, whether resulting in buffer overflow or integer overflow, this can result in crashes. These overflows also present the chance of arbitrary code execution or the injection of malicious media. Since this process involves the handling of potentially untrusted content from different sources it can be a primary target for attackers. As stated, untrusted media files are handled from external sources that can be potentially crafted to exploit vulnerabilities within the subsystem. The decoder also interacts with outside streaming protocols. These streams can be potentially malformed and contain dangerous payloads or affect availability. If the input is improperly handled this exploitation can persist into other subsystems, further jeopardizing the system as a whole.

Output Subsystem

The Output Subsystem in VLC Media Player is responsible for rendering decoded streams into visual and auditory outputs that the user can interact with. This includes both video output modules like OpenGL, Direct3D, or Vulkan, which render video frames to a display, and audio output modules like ALSA, PulseAudio, or CoreAudio, which process and send audio streams to speakers or headphones. The subsystem is designed to support a wide variety of hardware configurations and output formats, making it versatile across platforms. While it is less prone to critical security vulnerabilities compared to subsystems like the Decoder or Input, the Output Subsystem can still be impacted by bugs or resource handling errors. For instance, a flaw in memory management could cause resource leaks that degrade system performance, particularly in scenarios with multiple output devices (e.g., dual monitors or surround sound setups).

Interface Subsystem

The Interface Subsystem serves as the user's main interaction point with VLC Media Player. It includes various components such as the Graphical User Interface (GUI), Command-Line Interface (CLI), and web-based interfaces. This subsystem enables users to perform tasks like opening media files, controlling playback, configuring settings, and managing playlists. Its versatility ensures compatibility across platforms, ranging from desktop operating systems to mobile devices.

Security risks in the Interface Subsystem arise primarily from user inputs, especially in web-based controls. Insufficient input validation or sanitization here could expose the application to injection attacks, where malicious commands or scripts are executed within the system. For example, a poorly secured web interface could be vulnerable to cross-site scripting (XSS) or command injection, allowing an attacker to manipulate VLC's functionality or access sensitive data. Privilege escalation is another risk, where an attacker might exploit weaknesses in the subsystem to gain unauthorized access to higher-level system resources. Although the Interface Subsystem is not directly involved in media processing, flaws here can act as an entry point for attackers, leading to exploitation of other components such as the Input or Decoder Subsystems.

The Interface Subsystem plays a crucial role in usability and accessibility, but its interaction with untrusted inputs requires robust security measures like strict input validation, user authentication for remote access, and isolation of web interfaces to reduce attack surface. Given the subsystem's direct connection to the user and other internal components, maintaining its integrity is essential for both security and user experience.

Networking Subsystem

The Networking Subsystem is the backbone of VLC's ability to handle internet and network-based media streams. It supports a broad array of protocols, from secure options like HTTPS to less secure legacy ones such as FTP. This subsystem handles media buffering, data packet management, and secure communication using SSL/TLS. Despite its robustness, the networking subsystem poses risks if improperly implemented. For instance, errors in SSL/TLS validation could open VLC to man-in-the-middle (MITM) attacks, compromising secure connections. Additionally, support for legacy protocols like FTP could make VLC susceptible to sniffing attacks or unauthorized access. Poorly sanitized network inputs also increase the risk of injection attacks, which could corrupt system memory or lead to service disruptions.

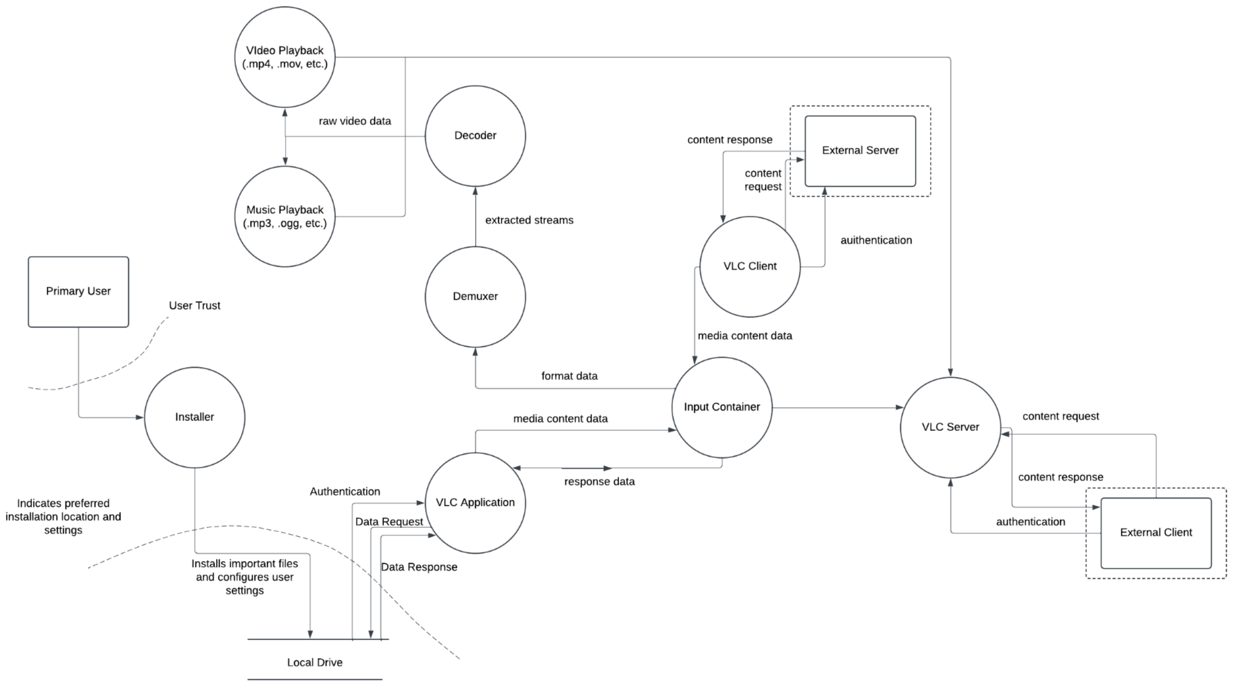
Architecture Analysis

Certain subsystems, such as the demuxer, decoder, and interface subsystems appear to be more susceptible to code level vulnerabilities. The demuxer subsystem has many different file types it must be able to manage, which can be susceptible to seemingly simple developer mistakes such as incorrectly checking bounds or “off by one” errors. The decoder subsystem is similar to the demuxer subsystem, however, this subsystem must actually decode the information encoded in the file provided. VLC supports many different file types, including proprietary file types which can have little and/or obscure documentation, which can make it very easy for developers to make assumptions about the file format, which can lead to mistakes and improper decoding that an attacker could take advantage of. The interface subsystem includes a GUI and CLI, which are very prone to developer mistakes in mishandling of certain commands and/or improper sanitization of user input.

Security is built into some of the subsystems, however, overall there is little distrustful decomposition. If one subsystem is compromised, the security of the entire local system could be called into question. For example, if the user can bypass the input subsystem's security and input a crafted malformed file, the demuxer and decoder subsystems will still attempt to decode the data, passing the exploit through the system.

External systems such as network-based media streams and plug-ins increase the attack surface of VLC. Network-based streams offer an opportunity not just for code-level errors but for confidentiality errors such as sniffers or man-in-the-middle attacks. Plug-ins are code allowed to run within VLC but not written by the developers of VLC, which can make them an easy target for attackers as the plugins may have vulnerabilities due to the plug-in developers not having the same knowledge of the system and experience as the VLC developers.

Threat Model



Machine Trust Boundaries:

These are essentially just the boundaries between a VLC client and external servers, or the boundary between VLC servers and external clients. This definitely varies from deployment to deployment since sometimes VLC runs as a client, other times a server (and each client/server will have a unique set of external actors based on the use case). VLC can also run entirely locally as an application and there are no relevant machine trust boundaries.

Other Trust Boundaries:

There is a local trust boundary that resides between the VLC application and the local drive. Unless specifically using VLC only for its client capabilities, this is an essential interaction. The media from a user's local drive is the key data processed by the application. This trust boundary is formed due to varying file permissions depending on the user's system. Without these assets, the application has much more limited functionality.

Assets to Threat Model Tracing

Table 2: Mapping VLC Assets to Threat Model Elements

Assets	**Threat Model Primitives**

Media files (e.g., `.mp4`, `.mkv`, `.avi`)	Local Drive, Input Container, External Server
User settings (e.g. .vlcrc file)	Local Drive
Codecs used by VLC for decoding media / Parsing multiple file types	Decoder
Streaming to other devices	VLC Server
Accepting streams from other devices	VLC Client
Plugins	Plugins (External Interactor)
Cross-platform availability (Live Servers)	VLC Server
Open-source code	VLC Application
Availability	VLC Client/Server

The location within architecture has an effect on the p(exploit) of an asset. Assets exposed to external interactions have a higher p(exploit) because of less control over the environment it previously resided in or was sourced from, leading to increased attack surface. Due to vlc's ability to stream to external devices and accept different streaming protocols, these interactions are exposed to man in the middle attacks. These assets are at increased risk of being intercepted and changed when being transferred. This directly affects the confidentiality and integrity of the media being streamed. Assets that are internal only have less exposure outside of our system's control. Although this limits the p(exploit) of the asset, indirectly attacking assets through file parsing, possible configuration manipulation, or memory mismanagement can still put these assets at risk.

No areas of our system have no assets, although there are locations in architecture where we specifically didn't identify assets which could leave certain vulnerabilities unaddressed. For file access and audio playback operations, vlc interacts with system level API. Although we did identify media files and codecs as assets, the APIs that parse these file paths, manage memory, and render audio can be potential attack vectors. We addressed the streaming features of vlc as assets, but the encryption and resulting handshake operation within the protocols were not specifically listed. These can bring concerns of improper token validation or encryption, exposing the system to man in the middle attacks

Certainly, there are a lot of assets that reside within the context of a VLC client or server. This is somewhat of a necessity as any client/server will be interacting with more untrusted, external actors, and as a result make up a larger percentage of the attack surface. Many of these assets are primarily at risk to availability in terms of CIA more than the other parts of the triangle. Assets that reside outside of the client/server primitives are generally of a lower priority due to the open-source nature of the application (source code, codecs). An exception to this is the local files (media, config) which is one of the most important assets in the system. Overall, it's very subjective whether or not the client and server contain too many assets, since a majority of the assets stem from the fact that they are client and servers in the first place.

Sources

"The Team That Powers VLC." *Increment*, https://increment.com/teams/the-team-that-powers-vlc/. Accessed 26 Nov. 2024.

VideoLAN. "VLC Security Advisories." *VideoLAN*, https://www.videolan.org/security/. Accessed 26 Nov. 2024.

VideoLAN. *VLC Repository*. GitHub, https://github.com/videolan/vlc. Accessed 26 Nov. 2024.

VideoLAN. *Commits to VLC 3.0.21*. Code.videolan.org, https://code.videolan.org/videolan/vlc/-/commits/3.0.21/?ref_type=tags. Accessed 26 Nov. 2024.

https://stackoverflow.com/questions/36220552/vlc-video-architecture

https://wiki.videolan.org/Hacker_Guide/