

libVLC

Security assessment by HashEye · prepared for OSTIF

HASHEYE AUDITED

PROJECT	libVLC
CLIENT	OSTIF
CATEGORY	Blockchain
PUBLISHED	May 1, 2025
REPORT ID	research-libvlc-2025-05-01-9j3c0p

This report was produced under HashEye's layered review process – **automated detection**, **pattern correlation**, and **senior manual verification** – with every finding signed off by a human reviewer. Full findings detail and on-chain attestation are available on the report page at hashey.io/audits/research-libvlc-2025-05-01-9j3c0p.

Prepared for: The VideoLAN Organization Organized by the Open Source Technology Improvement Fund, Inc.

Prepared by: William Woodruff and Facundo TUESCA

Table of Contents Table of Contents 1 Project Summary 3 Executive Summary 4 Project Goals 6 Project Targets 7 Project Coverage 8 Summary of Findings 9 Detailed Findings 11 1. Weak public key in self-update signature verification 11 2. Self-update APIs are HTTP-only 13 3. HTTP-only key lookup in self-update signature verification 15 4. Weak and attacker-controlled hash functions in self-update signature verification 17 5. Public key parameter malleability during self-updates 19 6. HTTP-only downloads during tool and contrib bootstrapping 21 7. Docker images are specified by tag instead of by digest 25 8. CI build might not be using the latest contribs prebuilt for MacOS 26 9. Insecure use of Docker CLI when authenticating to VideoLAN's container registry 28 10. Build dependency is downloaded from an out-of-date repository URL 29 11. Weak CSPRNG implementation on POSIX hosts 30 12. Weak CSPRNG implementation on OS/2 hosts 34 13. Weak public key in use of Sparkle framework on macOS 35 14. Outdated cryptographic dependencies 37 15. Weak cryptography in sftp module 39 16. File-based keystore uses default file mode 41 17. Weak TLS defaults in GnuTLS module 42 18. GnuTLS module susceptible to Logjam 44 19. Undocumented proxy server used for AcoustID lookups 46 20. VLC mirrors use plaintext rsync 48 21. Undefined behavior for specific Musicbrainz API responses 52 22. Address disclosure in dummy decoder 53 23. Pervasive address disclosure in logging components 54 24. Undefined behavior in SAT>IP port parsing 57

HashEye 1 OSTIF - libVLC

PUBLIC Security Assessment

A. Vulnerability Categories 60 B. Static Analysis Recommendations 61 B.1. Semgrep 61 B.2. clang-tidy 61 B.3. Docker image scanner 61 C. Fuzzing Harness Recommendations 62 C.1. Demuxer and Decoder Harness Recommendations 62 C.2. New Harness Recommendations 63 High-impact fuzzing harness candidates 63 Medium-impact fuzzing harness candidates 64 D. Fuzzer Corpus Recommendations 65 E. Code Quality Recommendations 67 About HashEye 69 Notices and Remarks 70

HashEye 2 OSTIF - libVLC

PUBLIC Security Assessment

Project Summary Contact Information The following project manager was associated with this project: Amanda Stickler, Project Manager amanda.stickler@hashey.io The following engineering director was associated with this project: William Woodruff, Engineering Director, Ecosystems william@hashey.io The following engineers were associated with this project: Facundo TUESCA, Senior Engineer Travis Peters, Principal Engineer facundo.tuesca@hashey.io travis.peters@hashey.io

William Woodruff, Engineering Director william@hashey.io Project Timeline The significant events and milestones of the project are listed below. Date Event February 18, 2025 Pre-project kickoff call March 14, 2025 Delivery of report draft May 9, 2025 Delivery of final draft May 14, 2025 Report readout meeting May 14, 2025 Delivery of final comprehensive report

HashEye 3 OSTIF - libVLC

PUBLIC Security Assessment

Executive Summary Engagement Overview OSTIF engaged HashEye to review the security of libVLC, the library and API components that drive the VLC media player. A team of three consultants conducted the review from February 18 to March 14, 2025, for a total of five engineer-weeks of effort. Our testing efforts focused on build hardening, susceptibility of libVLC to supply-chain attacks, fuzzer coverage and harness quality, as well as libVLC's own source code. With full access to public source code and documentation, we performed static and dynamic testing of libVLC and the VLC build system, using automated and manual processes. Where permitted by time and complexity, HashEye developed patches for findings and submitted them to the VLC maintainers for consideration.

Findings that were remediated during the report's editing period are marked as such. Additionally, where permitted by time and complexity, HashEye developed fuzzer dictionary improvements and additional fuzzing harnesses for libVLC APIs. These patches are indicated within their respective appendices. Due to the limited scope of the engagement as well as the size and complexity of the VLC project, this report does not make formal determinations of finding severity and difficulty. This report consists of two parts: a collection of findings made during manual and automated code review, and a set of appendices that document long-term improvements that the VLC community could make to improve the overall default security posture of libVLC and the VLC media player. Observations and Impact

We found multiple issues allowing an attacker to undermine or subvert the integrity of VLC and libVLC's builds (TOB-VLC-6, TOB-VLC-7, TOB-VLC-9, TOB-VLC-10, TOB-VLC-20) and self-update processes (TOB-VLC-1, TOB-VLC-2, TOB-VLC-3, TOB-VLC-4, TOB-VLC-5, TOB-VLC-13, TOB-VLC-20). We also found that VLC and libVLC largely rely on manual processes for dependency management across both builds and development infrastructure (such as Docker images), leaving the project susceptible to outdated and vulnerable dependencies. Recommendations Based on the findings identified during the security review, HashEye recommends that OSTIF take the following steps:

HashEye 4 OSTIF - libVLC

PUBLIC Security Assessment

- Remediate the findings disclosed in this report. These findings should be addressed as part of a direct remediation or any refactor that may occur when addressing other recommendations.
- Improve fuzzer coverage of core libVLC and decoder/demuxer components. Appendix C and Appendix D list recommendations for improvements that the VideoLAN maintainers can make to VLC's current fuzzing harnesses, fuzzing corpuses, as well as third-party distributed fuzzing campaigns (such as the oss-fuzz orchestrated campaign).
- Adopt automated source code and code quality analysis tooling. Appendix B lists recommendations for tools that can find problematic code and deployment (e.g. Docker image) patterns. These tools can be run as part of VLC's CI/CD processes in order to "gate" changes.

HashEye 5 OSTIF - libVLC

PUBLIC Security Assessment

Project Goals The engagement was scoped to provide a security assessment of the libVLC's build, supply-chain, deployment, and fuzzing practices. Specifically, we sought to answer the following non-exhaustive list of questions:

- Are libVLC (and related components) built with adequate compiler-side hardening flags. For example, do released Windows builds enable DEP, do Linux builds enable ASLR, etc.?
- Are libVLC (and related components) built in adequately isolated, hermetic, and reproducible CI/CD environments? For example, are build environments susceptible to cache poisoning, do they retrieve critical dependencies at runtime without integrity checks, etc.?
- Do libVLC (and related components) make use of tools and/or libraries that are unmaintained, or of unclear maintenance status?
- Do libVLC (and related components) make use of tools and/or libraries that have no clear canonical (or "true") source, or otherwise have unclear source provenance?
- Do libVLC (and related components) make use of significantly outdated tools and/or libraries, and do those components have known vulnerabilities?
- Are libVLC's notable or high-risk API surfaces (such as decoders, demuxers, and network protocol implementations) adequately represented within both in-tree and out-of-tree fuzzing harnesses?
- Are libVLC's fuzzing harnesses providing adequate coverage to notable or high-risk API surfaces?

HashEye 6 OSTIF - libVLC

PUBLIC Security Assessment

Project Targets The engagement involved reviewing and testing the following: VLC

Repository <https://code.videolan.org/videolan/vlc> Version f8cd3e546f054f3ddb8c3f2d7260c7b7f5591cf0
Type VLC media player Platform Multi-platform (macOS, Linux, Windows, BSD, Android, etc.)

docker-images

Repository <https://code.videolan.org/videolan/docker-images> Version 85396988dbad3c72a33ba5142eb6335e659c1d4e Type VideoLAN docker images Platform Multi-platform (macOS, Linux, Windows, BSD, Android, etc.)

HashEye 7 OSTIF - libVLC

PUBLIC Security Assessment

Project Coverage This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- Manual review of the targets listed above, including source review of the core libVLC and module source trees as well as source review of the VLC build system, GitLab CI/CD configuration, and miscellaneous container and infrastructure components
- Use of Semgrep on the VLC repository
- Use of grype on the docker-images repository

Coverage Limitations Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. During this project, we were unable to perform comprehensive testing of the following system elements, which may warrant further review:

- Decoder and demuxer implementations. We conducted a high-level review of multiple decoders and demuxers, but time constraints prevented an extensive review of individual modules.
- Network-facing parsers. We conducted a high-level review of VLC's HTTP/D and other protocol implementations, but time constraints prevented an extensive review of each protocol.
- Playback controls and state machine management. We conducted a high-level review of modules that exposed control APIs, but time constraints prevented an extensive review of VLC's playback control state machine.

HashEye 8 OSTIF - libVLC

PUBLIC Security Assessment

Summary of Findings The table below summarizes the findings of the review, including details on type, ID, Title, Type, Severity, and Impact.

ID	Title	Type	Severity	Impact
1	Weak public key in self-update signature verification	Cryptography	Weak	public key
2	Self-update APIs are HTTP-only	Cryptography	N/A	
3	HTTP-only key lookup in self-update signature verification	Cryptography	N/A	
4	Weak and attacker-controlled hash functions in self-update signature verification	Cryptography	Informational	
5	Public key parameter malleability during self-updates	Cryptography	Informational	
6	HTTP-only downloads during tool and contrib bootstrapping	Patching	N/A	
7	Docker images are specified by tag instead of by digest	Configuration	Informational	
8	CI build might not be using the latest contribs prebuilt for MacOS	Configuration	Informational	
9	Insecure use of Docker CLI when authenticating to VideoLAN's container registry	Data Exposure	N/A	
10	Build dependency is downloaded from an out-of-date repository	URL Configuration	Informational	
11	Weak CSPRNG implementation on POSIX hosts	Cryptography	N/A	
12	Weak CSPRNG implementation on OS/2 hosts	Cryptography	Informational	

HashEye 9 OSTIF - libVLC

PUBLIC Security Assessment

13	Weak public key in use of Sparkle framework on macOS	Cryptography	N/A	
14	Outdated cryptographic dependencies	Patching	N/A	
15	Weak cryptography in sftp module	Cryptography	N/A	
16	File-based keystore uses default file mode	Access Controls	N/A	
17	Weak TLS defaults in GnuTLS module	Cryptography	N/A	
18	GnuTLS module susceptible to Logjam	Cryptography	N/A	
19	Undocumented proxy server used for AcoustID lookups	Data Exposure	Informational	
20	VLC mirrors use plaintext rsync	Authenticatio n	N/A	
21	Undefined behavior for specific Musicbrainz API responses	Undefined Behavior	Informational	
22	Address disclosure in ddummy decoder	Data Exposure	Informational	
23	Pervasive address disclosure in logging components	Data Exposure	N/A	
24	Undefined behavior in SAT>IP port parsing	Undefined Behavior	N/A	

HashEye 10 OSTIF - libVLC

PUBLIC Security Assessment

Detailed Findings

1. Weak public key in self-update signature verification Type: Cryptography Finding ID: TOB-VLC-1 Target: include/vlc_pgpkey.h

```

Description In the absence of strong encryption via HTTPS (TOB-VLC-2), libVLC's self-
update APIs perform ad-hoc signature verification during both stages of the self-update flow.
During both stages, VLC retrieves an adjacent PGP .asc signature file and verifies it using
routines defined in src/misc/update_crypto.c. This signature verification is done against an
embedded PGP public key, defined in include/vlc_pgkey.h: /* gpg --export --armor "<id>" | sed -e
s/^\// -e s/\$/\\n"/ */ static const uint8_t videolan_public_key[] = { "-----BEGIN PGP PUBLIC
KEY BLOCK-----\n" "Version: GnuPG v2.0.19 (GNU/Linux)\n" "\n"
"mQGibBFD9w2QRBAcoEzH9KKirWE4wgiuPPynNnxks+p+t5i1z3CG+1XhagmTHo0f3\n"
"v8i19kKHV6WnVMn2CKJfGwTTLyX0JTrBM/4ABVtu11cHeeueeo+pCSkdoLzYJ5QF\n"
"HbByB6j33QubwKF0frEs+ge4LxzvYyCDAmNAW560Qt0AR9Lk1Fo5B1GXzwCg1kDk\n"
"RkSe7E0Znm1U2rYAQ2VPrfsEAIHr4oo0yUByPR7XpoD0KoaXEG0hjggh46lbgse+\n"
"dQx8YrxS9vXQLwYokfWlrs55avx9Ys0iVv2TMv7X4Tn5sTVaK5K+NbKhxhLORxGI\n"
"sgKqRn7W5SG5xo00w/dmQj756ppjITGbxjFuhYE0X5S6NeMhUuFci7sJ42R7F1Ko\n"
"6sYuA/w0MUxck4X0XeQf16ApyyenjE/UWbBNEhBmjEsZkYAFNc89pAEnEFSnIxK8\n"
"fcuQCQioM6ojjaW+aEs/q3/kLI0nat9LMLhNSCebjriMHwJDU70NeCn4nPWsfItT1\n"
"eKvbHNcX+3bq3D/i2Wa3PZ5YFFF01C61dHmVC9Ygh4sA0X009LQjVmLkZW9MQU4g\n"
"UmVsZWFzZSBTaWduaW5nIEtleSAoMjAxMymIaAQTEQIAKAUCUP3DZAibAwUJAFJX\n"
"AAYLcQgHAwIGFQgCCQoLBBYCAwECHgECF4AACgkQcYBx0+WNgtwQQwCg1JX6dDn5\n"
"gmPv4oczKpwlj5no0QwAn0HdT0xfmefXNQF1x+6t9BxBYXrNiEYEEBECAAYFALD9\n"
"xH4ACgkQp0FUn1ntKYdB+gCfS641cDBN2r0Kf/+Fra/p3bXgAeAAn2sJtSdN07Dv\n"
"rZeDWEbkhT620YOS\n" "=Npzf\n"
"-----END PGP PUBLIC KEY BLOCK-----\n" };

```

Figure 2.1: VLC's embedded PGP signing key This embedded key is a DSA 1024 key. DSA 1024 is considered to be roughly equivalent to 80-bit symmetric cryptography, which is considered excessively weak and within the reach of moderately-resourced attackers (circa 2015).

HashEye 11 OSTIF - libVLC

PUBLIC Security Assessment

Exploit Scenario An attacker who is able to crack VLC's embedded signing key can forge self-update responses, resulting in a fully compromised self-update process. In particular, for Windows hosts, an attacker who is able to crack the signing key can immediately achieve code execution on all VLC installations that perform self-updating, as self-updates are executed after being downloaded.

Recommendations Rotate to a suitably modern (i.e., non-DSA) and strong (margin equivalent to at least 128 bits of symmetric security) embedded signing key, and discontinue all use of the current DSA 1024 key. In particular, we recommend EdDSA (with P-256 or larger) or Ed25519 as a suitable replacement. If continuing to use DSA is an absolute necessity, we strongly recommend that VLC rotate to a DSA 3072 or stronger key. However, support for DSA keys larger than 1024 bits is not widespread.

References • Filippo Valsorda - DSA Is Past Its Prime • Weak Diffie-Hellman and the Logjam Attack

HashEye 12 OSTIF - libVLC

PUBLIC Security Assessment

2. Self-update APIs are HTTP-only Type: Cryptography Finding ID: TOB-VLC-2 Target: src/misc/update.c

Description libVLC contains public APIs that allow clients (including the VLC GUI) to automatically check for new releases of VLC. These APIs use a two-stage mechanism: 1. The first stage (GetUpdateFile) retrieves an "update file," which contains metadata about the latest version and a URL that points to it. 2. If the current version is outdated, the second stage (update_DownloadReal) downloads from the URL and, on Windows, prompts the user to exit and spawn the new VLC version. Both stages of this mechanism are HTTP-only. The first stage uses hard-coded base URLs: #ifndef NDEBUG # define UPDATE_VLC_STATUS_URL "http://update-test.videolan.org/vlc/status" #else # define UPDATE_VLC_STATUS_URL "http://update.videolan.org/vlc/status" #endif

Figure 1.1: HTTP base URLs for update files

Subsequently, the update file contains HTTP URLs as well. For example, here is a recent response from one of the above URLs:

3.0.20 http://get.videolan.org/vlc/3.0.20/vlc-3.0.20.tar.xz VideoLAN and the VLC development team present VLC 3.0.20 "Vetinari". VLC 3.0.20 is a medium update to VLC 3.0 branch: it updates codecs, activates AV1 hardware decoding, adds SuperResolution Scaling in hardware, fixes a FLAC issue and fixes playback of numerous formats. It also fixes a freeze when using frame-by-frame actions, and a few minor security issues. 3.0.20 adds a few fixes over 3.0.19, notably for crashes and security

issues. Figure 1.2: Update file, containing HTTP URL to final download Because both stages allow (and use) HTTP-only URLs, each is vulnerable to a man-in-the-middle attack, e.g., by an attacker in control of a router somewhere along the edge between the client machine and VLC's servers.

HashEye 13 OSTIF - libVLC

PUBLIC Security Assessment

VLC's self-update APIs attempt to mitigate this risk by performing ad-hoc PGP signature verification on both the update file and the download it references. However, as noted in TOB-VLC-1, this signature check is potentially insufficient to ensure the authenticity of responses. Exploit Scenario A victim who invokes these APIs on an untrusted network (such as a hotel, airport, or public network) can have their HTTP-only traffic intercepted and modified by an attacker with network-level access. The attacker can rewrite the update file response to point to any URL under their control, at which point VLC will retrieve an attacker-controlled update instead of an authentic one. On Windows machines the attacker can furthermore execute their malicious update immediately, assuming the user accepts the update dialog option. On Linux and other hosts the update is a source tarball and therefore cannot be executed directly. As mentioned above, the self-update APIs also perform signature verification. However, as noted in TOB-VLC-1, this verification uses insufficiently strong cryptography to defend against a moderately-resourced attacker. Recommendations Switch from HTTP to HTTPS URLs in both stages of the self-update flow, and enforce the presence of HTTPS. This finding was remediated with videolan/vlc!7157.

HashEye 14 OSTIF - libVLC

PUBLIC Security Assessment

3. HTTP-only key lookup in self-update signature verification Type: Cryptography Finding ID: TOB-VLC-3 Target: src/misc/update.c, src/misc/update_crypto.c

Description Per TOB-VLC-1, libVLC's self-update APIs perform signature verification on responses during both stages of the self-update flow. This signature verification is typically performed against the embedded key. However, if the signature's key ID does not match the embedded key, the self-update APIs attempt to retrieve the missing key from VideoLAN's servers: `if(memcmp(sign.issuer_longid, p_update->p_pkey->longid , 8) != 0) { msg_Dbg(p_update->p_libvlc, "Need to download the GPG key"); public_key_t *p_new_pkey = download_key(VLC_OBJECT(p_update->p_libvlc), sign.issuer_longid, videolan_public_key_longid);` Figure 3.1: Retrieval of a missing key The `download_key` helper performs an HTTP-only request based on the key's ID: `public_key_t *download_key(vlc_object_t *p_this, const uint8_t *p_longid, const uint8_t *p_signature_issuer) { char *psz_url; if(asprintf(&psz_url, "http://download.videolan.org/pub/keys/%.2X%.2X%.2X%.2X%.2X%.2X%.2X%.2X.asc", p_longid[0], p_longid[1], p_longid[2], p_longid[3], p_longid[4], p_longid[5], p_longid[6], p_longid[7]) == -1) return NULL;` Figure 3.2: HTTP request for missing key Like with TOB-VLC-2, this is vulnerable to a man-in-the-middle attack, e.g., by an attacker in control of a router somewhere along the edge between the client machine and VLC's servers. libVLC's self-update APIs attempt to mitigate this risk by verifying that the retrieved key is signed by the embedded key, similar to the mitigation in TOB-VLC-2. However, per TOB-VLC-1, the embedded key is weak by modern cryptographic standards and provides insufficient security margins.

HashEye 15 OSTIF - libVLC

PUBLIC Security Assessment

Exploit Scenario Per TOB-VLC-2, a victim who invokes these APIs on an untrusted network can have their HTTP-only key lookup traffic intercepted and modified by an attacker with network-level access. The attacker can then interpose any signing key they propose, at which point VLC will verify responses from the self-update APIs using the attacker-controlled signing key instead of an authentic VLC-controlled one. The scenario assumes that the attacker is able to contrive a valid signature against the embedded key, as the attacker-controlled key is not considered trusted until successfully verified against the embedded key. However, per TOB-VLC-1, a sufficiently resourced attacker may be able to crack the embedded key. Recommendations Switch from HTTP to HTTPS URLs during key lookup, and enforce the presence of HTTPS. This finding was remediated with videolan/vlc!7157.

HashEye 16 OSTIF - libVLC

PUBLIC Security Assessment

4. Weak and attacker-controlled hash functions in self-update signature verification Type: Cryptography Finding ID: TOB-VLC-4 Target: src/misc/update.c, src/misc/update_crypto.c

Description Per TOB-VLC-3, libVLC's self-update APIs will attempt to retrieve a missing key from VideoLAN's servers during digital signature verification. To establish trust in the new key, the self-update APIs attempt to verify the new key's signature against the embedded key: `uint8_t *p_hash = hash_from_public_key(p_new_pkey); if(!p_hash) { msg_Err(p_update->p_libvlc, "Failed to hash signature"); free(p_new_pkey); FREENULL(p_update->p_pkey); goto error; }`

`if(verify_signature(&p_new_pkey->sig, &p_update->p_pkey->key, p_hash) == VLC_SUCCESS) { free(p_hash); msg_Info(p_update->p_libvlc, "Key authenticated"); free(p_update->p_pkey); p_update->p_pkey = p_new_pkey; } else { free(p_hash); msg_Err(p_update->p_libvlc, "Key signature invalid !"); goto error; }`

Figure 4.1: Verifying the untrusted key against the embedded, trusted key As part of this signature verification, libVLC's self-update APIs calculate a hash over the new key within the `hash_from_public_key` helper. The hash function used to calculate this hash is derived solely from the new (i.e., untrusted) key:

HashEye 17 OSTIF - libVLC

PUBLIC Security Assessment

```
/* * Generate a hash on a public key, to verify a signature made on that hash * Note that we need the signature (v4) to compute the hash */ uint8_t *hash_from_public_key( public_key_t *p_pkey ) {  
/* omitted for brevity */
```

```
error = gcry_md_open( &hd, p_pkey->sig.digest_algo, 0 );
```

Figure 4.2: Unverified key controls hash function selection This produces unnecessary malleability in the signature verification check, as an attacker can introduce an extremely weak cryptographic hash (such as MD4, MD5, or SHA-1) in an attempt to simplify a signature forgery attack. In practice this signature forgery is largely thwarted by the infeasibility of preimage attacks, even on weak cryptographic hash functions. As such, this is an informational finding. Recommendations Short term, VLC should perform a rotation of the embedded key (per TOB-VLC-1) and include a suitably modern, strong general-purpose hash function for use with the new signing key. In particular, we recommend choosing a hash function from the SHA-2 or SHA-3 family, with SHA-2/256 as a conservative baseline. Moreover, the key verification flow should be updated so that the embedded key always controls the hash algorithm used. Long term, per TOB-VLC-5, VLC should transition away from ad-hoc handling of cryptographic primitives via low-level gcrypt APIs. Instead, we recommend that VLC apply a high-level, misuse-resistant cryptography library like libsodium to key handling and digital signature operations.

HashEye 18 OSTIF - libVLC

PUBLIC Security Assessment

5. Public key parameter malleability during self-updates Type: Cryptography Finding ID: TOB-VLC-5 Target: src/misc/update_crypto.c

Description During online key lookup, libVLC's self-update APIs parse and validate RSA and DSA public keys prior to chaining them against the embedded key. This occurs in

```
parse_public_key_packet: static int parse_public_key_packet( public_key_packet_t *p_key, const uint8_t *p_buf, size_t i_packet_len ) { if( i_packet_len < 6 ) return VLC_EGENERIC;
```

```
size_t i_read = 0;
```

```
p_key->version = *p_buf++; i_read++; if( p_key->version != 4 ) return VLC_EGENERIC;
```

```
/* XXX: warn when timestamp is > date ? */ memcpy( p_key->timestamp, p_buf, 4 ); p_buf += 4; i_read += 4;
```

```
p_key->algo = *p_buf++; i_read++; if( p_key->algo == GCRY_PK_DSA ) { READ_MPI( p_key->sig.dsa.p, 3072 ); READ_MPI( p_key->sig.dsa.q, 256 ); READ_MPI( p_key->sig.dsa.g, 3072 ); READ_MPI( p_key->sig.dsa.y, 3072 ); } else if ( p_key->algo == GCRY_PK_RSA ) { READ_MPI( p_key->sig.rsa.n, 4096 ); READ_MPI( p_key->sig.rsa.e, 4096 ); } else return VLC_EGENERIC;
```

```
if( i_read == i_packet_len ) return VLC_SUCCESS;
```

```
/* some extra data eh? */
```

```
error: return VLC_EGENERIC; } Figure 5.1: RSA and DSA public key parsing
```

HashEye 19 OSTIF - libVLC

PUBLIC Security Assessment

This parsing routine performs insufficient verification of the public parameters in both RSA and DSA keys. For RSA:

- The public modulus (n) should be restricted to an appropriate security margin (2048

- as a minimum, but ideally 3072 or higher).
- The public exponent (e) should be restricted to a safe exponent. 65537 is the standard choice.

For DSA:

- The parameters p , q , g , and public key y should be cross-validated using the process defined in NIST SP 800-89, Section 5.3.1 (“Explicit Full Public Key Validation for DSA”). However, per TOB-VLC-1, we separately recommend that VLC entirely rotate away from use of DSA public keys or signatures. The above is not immediately exploitable, as an attacker would have to first break the embedded key mentioned in TOB-VLC-1 or otherwise perform a signature forgery. Consequently, we consider this an informational finding. Recommendations Perform the public key parameter validation steps described above. Long term, we recommend that VLC transition away from ad-hoc parsing of PGP key and signature packets, and transition to a more modern, misuse-resistant set of APIs and cryptographic primitives. In particular, we recommend libsodium for misuse-resistant keypair generation and digital signatures in C. References
- NIST SP 800-78-5

HashEye 20 OSTIF - libVLC

PUBLIC Security Assessment

6. HTTP-only downloads during tool and contrib bootstrapping Type: Patching Finding ID: TOB-VLC-6 Target: extras/tools/tools.mak, contrib/src/main.mak

Description VLC’s build contains multiple bootstrapping phases. Two of these phases (the “tools” and “contrib” phases) make extensive use of HTTP-only (i.e., unencrypted and unauthenticated) network requests to retrieve non-vendored build and runtime dependencies. These requests may be subject to tampering by an attacker with network-level access, resulting in compromised builds.

In both phases, dependencies are declared in Makefile syntax via .mak files, which are then included into a larger build that performs each dependency’s download, extraction, build, linkage, etc. The download component of each phase uses GNU Make’s call function to invoke the download function, which is defined conditionally depending on the presence of wget or curl.

```
ifeq ($(shell command -v curl >/dev/null 2>&1 || echo FAIL),) download = curl -f -L -- "$$(1)" > "$@.tmp" && touch $@.tmp && mv $@.tmp $@ else ifeq ($(shell command -v wget >/dev/null 2>&1 || echo FAIL),) download = rm -f $@.tmp && \ wget --passive -c -p -O $@.tmp "$$(1)" && \ touch $@.tmp && \ mv $@.tmp $@ else ifeq ($(shell command -v fetch >/dev/null 2>&1 || echo FAIL),) download = rm -f $@.tmp && \ fetch -p -o $@.tmp "$$(1)" && \ touch $@.tmp && \ mv $@.tmp $@ else download = $(error Neither curl nor wget found!) endif
```

Figure 6.1: Definition of download function in the tools phase

```
ifeq ($(shell curl --version >/dev/null 2>&1 || echo FAIL),) download = curl -f -L -- "$$(1)" > "$@" else ifeq ($(shell wget --version >/dev/null 2>&1 || echo FAIL),) download = (rm -f $@.tmp && \ wget --passive -c -p -O $@.tmp "$$(1)" && \ touch $@.tmp && \
```

HashEye 21 OSTIF - libVLC

PUBLIC Security Assessment

```
mv $@.tmp $@ ) else ifeq ($(command -v fetch >/dev/null 2>&1 || echo FAIL),) download = (rm -f $@.tmp && \ fetch -p -o $@.tmp "$$(1)" && \ touch $@.tmp && \ mv $@.tmp $@) else download = $(error Neither curl nor wget found) endif
```

Figure 6.2: Definition of download function in the contrib phase

Calls to download are done for both external resources and VLC-hosted resources, including “prebuilt” contrib bundles that contain a compiled set of runtime dependencies. These “prebuilt” bundles shortcut VLC’s lengthy bootstrapping process by removing the need to compile dozens of runtime dependencies from scratch:

```
PREBUILT_URL=http://download.videolan.org/pub/videolan/contrib/$(HOST)/vlc-contrib-$(HOST)-latest.tar.bz2
```

```
vlc-contrib-$(HOST)-latest.tar.bz2: $(call download,$(PREBUILT_URL))
retrieval PREBUILT_URL is additionally overridden (via VLC_PREBUILT_CONTRIBS_URL) in VLC's GitLab
CI/CD configuration to point to a different subdomain (artifacts.videolan.org instead of
download.videolan.org). Both phases use a mosaic of HTTP and HTTPS requests: some dependencies are
requested via HTTPS, while others use HTTP explicitly and appear to exist on hosts that offer no
HTTPS upgrade route. For example, the live555 contrib comes from http://live555.com , which does
not appear to support HTTPS: LIVE555_VERSION := 2022.07.14 LIVE555_FILE :=
live.$(LIVE555_VERSION).tar.gz LIVEDOTCOM_URL :=
http://live555.com/LiveMedia/public/$(LIVE555_FILE)
```

```
ifdef BUILD_NETWORK ifdef GNUV3 PKGS += live555 endif endif
```

```
ifeq ($(call need_pkg,"live555"),) PKGS_FOUND += live555 endif
```

```
$(TARBALLS)/$(LIVE555_FILE): $(call download_pkg,$(LIVEDOTCOM_URL),live555)
```

HashEye 22 OSTIF - libVLC

PUBLIC Security Assessment

.sum-live555: \$(LIVE555_FILE) Figure 6.4: HTTP-only contrib retrieval for live555 VLC's bootstrapping phases attempt to mitigate the lack of consistent transport security via the presence of SHA512SUMS files for each contrib or tool dependency, which are included with VLC's own source distribution. However, per TOB-VLC-20, the trustworthiness of the SHA512SUMS files are themselves subject to weaknesses in VLC's mirror distribution scheme. Moreover, not all downloads have a corresponding SHA512SUMS file. Notably, "prebuilt" contrib distributions lack a corresponding SHA512SUMS, as prebuilt contribs represent a "rolling" build state that would require continuous updates to the source tree. In summary: downloads during the tool and contrib bootstrapping phases do not consistently use HTTPS for transport integrity and security, even when hosts being downloaded from support HTTPS. Both phases additionally use SHA-512 checksum verification to prevent network tampering, but the "prebuilt" contrib flow bypasses this checksum. Exploit Scenario An attacker on the network path between a VLC builder (such as an end user, Linux distribution, or the official VLC CI/CD) may be able to maliciously manipulate the contexts of dependencies downloaded during the bootstrapping phase, resulting in a compromised build of VLC or libVLC. Under ideal conditions, this attacker would be stymied by the presence of SHA512SUMS files within the VLC source distribution. However, per TOB-VLC-20, the contents of these checksum files are themselves subject to manipulation. Additionally, in dependency-update scenarios (e.g. updates by the VideoLAN maintainers themselves), no prior trustworthy digest is available for comparison. Recommendations Update as many tool and contrib dependency downloads to HTTPS as possible. Where doing so is impossible (e.g., due to origins that are HTTP only), we recommend that VLC upload a copy to its own HTTPS-secured origin (e.g., download.videolan.org) and exclusively use that copy during bootstrapping. Additionally, as a misuse/regression resistance mechanism, we recommend that VLC's build system check that supplied URLs are HTTPS-only, use suitably modern versions of TLS (1.2 or newer), and do not perform insecure redirects back to HTTP. For the curl case of the download function, this can be accomplished by always passing the following arguments: • --proto 'https': force HTTPS, forbid all other protocols (including in redirects)

HashEye 23 OSTIF - libVLC

PUBLIC Security Assessment

- --tlsv1.2: force TLS 1.2 or newer Longer term, we recommend that the VideoLAN maintainers consider a more structured approach to dependency management during the tool and contrib phases, one that includes integrity and authenticity as a core property of the management system itself. Potential management systems include vcpkg, Conan, and Nix. This finding was remediated with videolan/vlc!7157.

HashEye 24 OSTIF - libVLC

PUBLIC Security Assessment

7. Docker images are specified by tag instead of by digest Type: Configuration Finding ID: TOB-VLC-7 Target: extras/ci/gitlab-ci.yml, files in docker-images repository

Description VLC's CI uses Docker images specified by tag. This applies to the images hosted in VLC's registry and the base images used in all Dockerfile definitions in the

https://code.videolan.org/videolan/docker-images repository. Tags are not static, and users with write access to the registry can upload a new image and rewrite an existing tag to point to it. This has implications for reproducibility and security since builds will silently start using the new image. Using digests instead of tags prevents this attack vector since the image referred to by the digest will always be the same. VLC_WIN64_IMAGE: registry.videolan.org/vlc-debian-win64-posix:20241118101328 VLC_WIN_LLVM_MSVCRT_IMAGE: registry.videolan.org/vlc-debian-llvm-msvcrt:20241118101328 VLC_WIN_LLVM_UCRT_IMAGE: registry.videolan.org/vlc-debian-llvm-ucrt:20241118101328 VLC_DEBIAN_IMAGE: registry.videolan.org/vlc-debian-unstable:20241112155431 VLC_ANDROID_IMAGE: registry.videolan.org/vlc-debian-android:20241118101328 VLC_SNAP_IMAGE: registry.videolan.org/vlc-ubuntu-focal:20231013031754 VLC_RASPBIAN_IMAGE: registry.videolan.org/vlc-ubuntu-raspberrypi:20240806085528 VLC_WASM_EMSCRIPTEN: registry.videolan.org/vlc-debian-wasm-emscripten:20250207201514

Figure 7.1: Images downloaded by tag Recommendations Change the CI builds and Dockerfiles to use digests rather than tags.

HashEye 25 OSTIF - libVLC

PUBLIC Security Assessment

8. CI build might not be using the latest contribs prebuilt for MacOS Type: Configuration Finding ID: TOB-VLC-8 Target: extras/ci/get-contrib-sha.sh

Description VLC builds in CI use pre-built contrib packages downloaded from <https://artifacts.videolan.org>. These packages are stored with a filename containing the commit hash used to build them (e.g., vlc-contrib-aarch64-apple-darwin19-28b601318640dc3be3b463fbb39e480179a-d8ebe.tar.bz2). The CI build chooses which contrib archive to download by using git to check for changes in certain folders, and choosing the commit that contains the latest change to them. However, the folder checked for macOS builds is extras/package/macos, which does not exist (the correct folder name is macosx). This means CI builds might not use the pre-built contrib packages with the latest changes when building the macOS artifacts. case \$1 in win32*|win64*|uwp*) VLC_CONTRIB_REBUILD_PATHS+=("extras/package/win32") ;; ios*|tvos*) VLC_CONTRIB_REBUILD_PATHS+=("extras/package/apple") ;; macos*) VLC_CONTRIB_REBUILD_PATHS+=("extras/package/macos") ;; raspbian*) VLC_CONTRIB_REBUILD_PATHS+=("extras/package/raspberrypi") ;; snap*) VLC_CONTRIB_REBUILD_PATHS+=("extras/package/snap") ;; wasm*) VLC_CONTRIB_REBUILD_PATHS+=("extras/package/wasm-emscripten") ;; debian*|android*) ;; *) VLC_CONTRIB_REBUILD_PATHS+=("extras/package") esac

HashEye 26 OSTIF - libVLC

PUBLIC Security Assessment

Figure 8.1: Platform-specific folders checked for changes when choosing the Git commit for downloading the pre-built contrib packages. Recommendations Change the folder name used in the extras/ci/get-contrib-sha.sh script to macosx, matching the folder name present in the repository. This finding was remediated with videolan/vlc!7159.

HashEye 27 OSTIF - libVLC

PUBLIC Security Assessment

9. Insecure use of Docker CLI when authenticating to VideoLAN's container registry Type: Data Exposure Finding ID: TOB-VLC-9 Target: .gitlab-ci.yml in the docker-images repository

Description For authentication using the Docker CLI, the --password-stdin option should be preferred over the -p option, to prevent the password from ending up in the shell's history or log files. script="\$script\ndocker login -u=\\$REGISTRY_LOGIN -p=\\$REGISTRY_AUTH registry-mgmt.videolan.org" script="\$script\ndocker push registry-mgmt.videolan.org/\\$d:\\$DATE"

Figure 9.1: Use of -p option when authenticating to the VLC container registry Exploit Scenario An attacker who obtains access to (non-public) run logs or shell histories from VLC's GitLab CI/CD may be able to recover the Docker registry credential used to push to the registry. From there, an attacker may be able to pivot to additional compromise of VLC's CI/CD or build artifacts by overwriting existing image tags, per TOB-VLC-7. Recommendations Use --password-stdin over -p, as shown in Figure 9.2: \$ echo \$REGISTRY_AUTH | docker login --username \$REGISTRY_LOGIN --password-stdin registry-mgmt.videolan.org

Figure 9.2: Example usage of --password-stdin A remediating patch for this finding was opened with videolan/docker-images!313 and was still under review at report finalization time.

HashEye 28 OSTIF - libVLC

PUBLIC Security Assessment

10. Build dependency is downloaded from an out-of-date repository URL Type: Configuration Finding ID: TOB-VLC-10 Target: docker-images/medialibrary-win32/Dockerfile, docker-images/medialibrary-win64/Dockerfile

Description The RapidJSON library is downloaded from the miloyip/rapidjson GitHub repository, which was transferred and now redirects to Tencent/rapidjson. The original owner identity (miloyip) could create a new repository with the same name and upload malicious code, without any restrictions despite the transfer and current redirect (see GH docs here). `cd /build/ml && wget -q https://github.com/miloyip/rapidjson/archive/v$RAPIDJSON_VERSION.tar.gz && \` Figure 10.1: Downloading RapidJSON release from miloyip/rapidjson repository The above is not immediately exploitable, as the Dockerfiles in question also check the integrity of the download via a pre-established SHA2/256 digest. However, subsequent updates may mistakenly assume that the pre-redirect identity is trusted and incorrectly trust any artifact digests. Consequently, we consider this an informational-only finding. Recommendations Replace the download URL with the current official repository, `https://github.com/Tencent/rapidjson`. A remediating patch for this finding was opened with videolan/docker-images!322 and was still awaiting review at report finalization time. References

- GitHub documentation: Transferring a repository

HashEye 29 OSTIF - libVLC

PUBLIC Security Assessment

11. Weak CSPRNG implementation on POSIX hosts Type: Cryptography Finding ID: TOB-VLC-11 Target: src/posix/rand.c

Description libVLC provides APIs for a cryptographically secure pseudorandom number generator (CSPRNG) via `include/vlc_rand.h`. These APIs are implemented in a platform specific manner, e.g., `src/posix/rand.c` for POSIX systems and `src/win32/rand.c` for Windows hosts. On POSIX systems, the CSPRNG APIs include a one-time random state initialization function, `vlc_rand_init`: `static uint8_t okey[BLOCK_SIZE], ikey[BLOCK_SIZE];`

```
static void vlc_rand_init (void) { uint8_t key[BLOCK_SIZE];  
  
/* Get non-predictible value as key for HMAC */ int fd = vlc_open ("/dev/urandom", O_RDONLY); if (fd == -1) return; /* Uho! */  
  
for (size_t i = 0; i < sizeof (key);) { ssize_t val = read (fd, key + i, sizeof (key) - i); if (val > 0) i += val; }  
  
/* Precompute outer and inner keys for HMAC */ for (size_t i = 0; i < sizeof (key); i++) { okey[i] = key[i] ^ 0x5c; ikey[i] = key[i] ^ 0x36; }  
  
vlc_close (fd); } Figure 11.1: Random state initialization with vlc_rand_init
```

HashEye 30 OSTIF - libVLC

PUBLIC Security Assessment

This initialization function fails silently if `/dev/urandom` cannot be opened for any reason. This in turn catastrophically compromises the initial random state, as the static `okey` and static `ikey` members have static storage duration and therefore are zero-initialized. Open `/dev/urandom` is unlikely to occur on ordinary Linux and other POSIX hosts under normal operation. However, `/dev/urandom` is not guaranteed to exist or be readable, e.g., due to file descriptor exhaustion attacks. Therefore consumers must handle its failure modes appropriately. Once initialized, `vlc_rand_bytes` on POSIX hosts uses HMAC-MD5 with a mixed-in counter and timestamp to provide a stream of random bytes: `void vlc_rand_bytes (void *buf, size_t len) { static vlc_mutex_t lock = VLC_STATIC_MUTEX; static uint64_t counter = 0; struct timespec ts;`

```
timespec_get(&ts, TIME_UTC);
```

```

while (len > 0) { uint64_t val; vlc_hash_md5_t mdi, mdo; uint8_t mdi_buf[VLC_HASH_MD5_DIGEST_SIZE];
uint8_t mdo_buf[VLC_HASH_MD5_DIGEST_SIZE];

vlc_hash_md5_Init (&mdi); vlc_hash_md5_Init (&mdo);

vlc_mutex_lock (&lock); if (counter == 0) vlc_rand_init (); val = counter++;

vlc_hash_md5_Update (&mdi, ikey, sizeof (ikey)); vlc_hash_md5_Update (&mdo, okey, sizeof (okey));
vlc_mutex_unlock (&lock);

vlc_hash_md5_Update (&mdi, &ts, sizeof (ts)); vlc_hash_md5_Update (&mdi, &val, sizeof (val));
vlc_hash_md5_Finish (&mdi, mdi_buf, sizeof(mdi_buf)); vlc_hash_md5_Update (&mdo, mdi_buf,
sizeof(mdi_buf)); vlc_hash_md5_Finish (&mdo, mdo_buf, sizeof(mdo_buf));

memcpy (buf, mdo_buf, (len < sizeof(mdo_buf)) ? len : sizeof(mdo_buf));

if (len < sizeof(mdo_buf)) break;

len -= sizeof(mdo_buf);

```

HashEye 31 OSTIF - libVLC

PUBLIC Security Assessment

buf = ((uint8_t *)buf) + sizeof(mdo_buf); } } Figure 11.2: CSPRNG implemented with HMAC-MD5 on POSIX hosts While not suitable for other cryptographic constructions, MD5 is not insecure for use in HMAC constructions. However, the entropy mixed into VLC's use of the HMAC-MD5 construction is non-ideal:

- timespec_get is used to mix a UTC timestamp into the HMAC. This may be fine-grained but is not guaranteed to be so, and is potentially guessable by an attacker who engages in timed network traffic with VLC.
- A counter is additionally mixed into the HMAC, preventing the scenario of two threads producing the same RNG state by obtaining the same timespec. This counter is locked and incremented atomically via libVLC's mutex APIs. However, doing this is both brittle (as it ties the soundness of the CSPRNG to a custom spinlock implementation) and slow (as lock acquisitions scale linearly with the size of the CSPRNG request).

Exploit Scenario An attacker who is able to induce an I/O failure during the opening of /dev/urandom is able to fatally compromise the initialization of the CSPRNG, leading to predictable random streams in all subsequent uses within VLC. This includes uses of the CSPRNG for secure network traffic (including RTP/RTSP), HTTP authentication nonces, as well as any indirect uses of the CSPRNG by Lua plugins. In particular, an attacker may be able to induce this failure via a file descriptor exhaustion attack, or by targeting (relatively marginal) POSIX hosts where /dev/urandom is not present. Separately, an attacker who identifies a weakness in libVLC's spinlock implementation may be able to induce duplicate CSPRNG states between multiple threads, compromising subsequent users of the CSPRNG.

Recommendations The following changes should be made to VLC's POSIX CSPRNG:

- Replace the use of /dev/urandom with dedicated CSPRNG APIs, such as getrandom(2) (Linux) and arc4random_buf (macOS, BSDs). The latter uniformly use modern, secure stream ciphers despite their name: macOS uses AES as of macOS 10.12, while the BSDs generally use ChaCha20. This simultaneously eliminates any risks involved in performing I/O on /dev/urandom and eliminates the need for a custom HMAC-MD5 construction, as the CSPRNG APIs are self-synchronizing and provide sufficiently high-quality random bytes.
- Eliminate the HMAC-MD5 construction entirely. While not insecure in this setting, the use of MD5 is non-ideal, and requires further non-ideal entropy sources (a

HashEye 32 OSTIF - libVLC

PUBLIC Security Assessment

synchronized global counter and guessable timestamps) to provide adequate per-thread randomness. Eliminating the HMAC-MD5 construction would both improve VLC's CSPRNG performance on POSIX hosts and reduce the amount of predictable input present during CSPRNG operation. The win32/rand.c implementation of vlc_rand_bytes is an ideal reference point for both of these recommendations, as it uses the system CSPRNG via BCryptGenRandom with no additional userspace constructions.

HashEye 33 OSTIF - libVLC

PUBLIC Security Assessment

12. Weak CSPRNG implementation on OS/2 hosts Type: Cryptography Finding ID: TOB-VLC-12 Target: src/os2/rand.c

```
Description On OS/2 hosts, libVLC's CSPRNG API (vlc_rand_bytes) provides non-cryptographic randomness: void vlc_rand_bytes (void *buf, size_t len) { QWORD qwTime; uint8_t *p_buf = (uint8_t *)buf;
```

```
while (len > 0) { DosTmrQueryTime( &qwTime );
```

```
*p_buf++ = ( uint8_t )( qwTime.uLlo * rand()); len--; } }
```

Figure 12.1: Non-cryptographically secure RNG on OS/S In particular: randomness is driven entirely by two predictable entropy sources: rand and DosTmrQueryTime . No other, less guessable sources of entropy are used. Moreover, no thread isolation or synchronization is performed, meaning that multiple threads may produce the same RNG output. We consider this an informational finding, as the status of VLC and libVLC on OS/2 is unclear. Similarly, the status of suitable CSPRNG API alternatives for OS/2 hosts is unclear, meaning that straightforward remediation may not be possible. Recommendations Use a strong, OS-provided CSPRNG on OS/2, if one is available. If no OS-provided CSPRNG API is offered by OS/2, then the vlc_rand_bytes should perform a best-effort secure seed initialization followed by a modern, secure stream cipher such as ChaCha20 as a DRNG.

HashEye 34 OSTIF - libVLC

PUBLIC Security Assessment

13. Weak public key in use of Sparkle framework on macOS Type: Cryptography Finding ID: TOB-VLC-13 Target: modules/gui/macosx/Resources/dsa_pub.pem

Description VLC uses the Sparkle framework to automate software updates on macOS, in a manner similar to the self-update APIs used on Linux and Windows hosts (per TOB-VLC-1, TOB-VLC-2, etc.).

As in TOB-VLC-1, macOS self-updates are signed and verified using a keypair controlled by the VideoLAN maintainers. The public half of this keypair is embedded into the macOS application distribution of VLC: -----BEGIN PUBLIC KEY-----

```
MIID0jCCAi0GByqGSM44BAEwggIgAoIBAQCkgUAUyKhUu7L6evTnJFIsI0LBRZ62
jtvT/DgPLOKmhZ/hDjnwyaid8RwrygmvnCUyd5Lca4bS8jWLELDdkKhNt8CshgdVV
6tPJ/EL0qSCuWd3+ttddT80j/HVA5nV2TpkbtdRsXJ4v3W676NqWuRFGV32zI23q
JSvs8sBAuTh6nxmtSjfnclLULwrhqitxNHGQvbLXdN2pgi/HzaJUz0Va5RHRMc
05kjJut3Y3NQFoDem6r/zTKYaXi0FgMhtKMDfFILvK/+TDQ8fXKvxK/+PegfbCFi
IUTmxelt0tsUae8i886AbBjnWYDj6v4Kitl5tCIh4VmRd4wvFKuZV6dNAhUAjAI
kpfUNsZ5N3CIQzvtZHVn5mUCggEAJZ8BB00ypJRYK3TrPz+wAg17pV0KhOT4HIVm
4J0c0iYLTbZTdfXx3PyRPWpw5VTdSq63LS04WkKhBZ5Ducdw122u78TGPuNAVBKq
co/0B1Xte3zWBG0gdCnutF9rrMYLAGox+g24KcnwLzCqXK0o6wq7MKw8/A6Kfvg
KHw6LufL/eFgYcvHZQ+Bj8zoRctscd+qP0duSjy/LoyoRYyTniGodEZ6Etv1Pk/K
lfZ02vzNmK0L/SVT9batb88HwFG9Xvp1AsMir5eshvLZ/XfeEhVfXM68QYRtG8pt
RpDuk6fsAseZLdYoJA8fsXdxvCCexXghVmMpfD+tg3BRcl+ptQOCAQUAAoIBABYz
qSJ+jJFka1gJ4i5KUKBLsYMwT3gF8YlZ3Ny6CsRjcdLlIOQdncUu+aRp8by43Gxw
03tqTEo9k6FkdVEourJ0dc3C3bH8atVYcw4w1APLbr4PVufciJaVXfDudKBSSvly
hpP+U6LD9T55LvBz9ebUgT1VjqWRU5Cq/7AgsI3nxH3/1oDD6cZBNnCW739iftm
KIL7UNJoVIIy+fvJZP7sx/f0EVluN55SqE8LQPgL6oYhgUBLTud00tchb049aMJs
po2zD5N0/Wf7gaDYba674R5sVAEY6yYe8SIXuntf5IPk5nL67fkjeDxvnnq3ern dEB9sijtTmSV2h1xUzXg= -----END
PUBLIC KEY-----
```

Figure 13.1: Embedded Sparkle Framework update signing key As with the self-update APIs, this is a DSA 1024 key. As such, it offers an insufficient security margin. Exploit Scenario As with TOB-VLC-1, an attacker who is able to crack VLC's embedded Sparkle signing key can forge self-update responses, resulting in a fully compromised self-update process on macOS hosts.

HashEye 35 OSTIF - libVLC

PUBLIC Security Assessment

Recommendations VLC currently depends on version 1.16.0 of Sparkle, which only supports DSA signatures. The VideoLAN maintainers should upgrade VLC to Sparkle 1.21 or later, which includes support for EdDSA keypairs and signatures. Following the upgrade, the VideoLAN maintainers should generate an EdDSA keypair and rotate out the current DSA keypair. Version 1.27 and later of Sparkle remove support for DSA entirely, retaining only EdDSA. More information can be found in Sparkle's EdDSA migration guide.

HashEye 36 OSTIF - libVLC

PUBLIC Security Assessment

14. Outdated cryptographic dependencies Type: Patching Finding ID: TOB-VLC-14 Target: contrib/src/gcrypt, contrib/src/gnutls, contrib/src/nettle, contrib/src/libtasn1

Description VLC's build includes "contrib" builds of dozens of dependencies, including core cryptographic libraries. Of these, we identify at least three with notable bugfixes/security advisories made since the last update: 1. libVLC currently depends on libgcrypt 1.10.1, which is approximately three years old (2022-03-28). Subsequent releases of gcrypt have fixed numerous security-relevant bugs per the NEWS file. In particular, libgcrypt 1.10.1 is vulnerable to CVE-2024-2236, which enables Bleichenbacher-style attacks against RSA encryption. GCRYPT_VERSION := 1.10.1 Figure 14.1: Outdated gcrypt in rules.mak 2. libVLC currently depends on gnutls 3.8.3, which is multiple releases behind the latest (3.8.9). This newer release fixes at least three security advisories currently present with 3.8.3: CVE-2024-28834, CVE-2024-28835, and CVE-2024-12243. GNUTLS_MAJVERSION := 3.8 GNUTLS_VERSION := \$(GNUTLS_MAJVERSION).3 Figure 14.2: Outdated gnutls in rules.mak 3. libVLC currently depends on nettle 3.9, which is multiple releases behind the current latest stable release (3.10.1). Subsequent releases of nettle have fixed a number of cryptographic and memory corruption bugs per the NEWS file. NETTLE_VERSION := 3.9 NETTLE_URL := \$(GNU)/nettle/nettle-\$(NETTLE_VERSION).tar.gz Figure 14.3: Outdated nettle in rules.mak 4. libVLC currently depends on libtasn1 4.19.0, which is behind the current latest stable release (4.20.0). This newer release fixes at least one CVE (CVE-2024-12133).

HashEye 37 OSTIF - libVLC

PUBLIC Security Assessment

LIBTASN1_VERSION := 4.19.0 LIBTASN1_URL := \$(GNU)/libtasn1/libtasn1-\$(LIBTASN1_VERSION).tar.gz Figure 14.4: Outdated libtasn1 in rules.mak Exploit Scenario An attacker on the network path during any VLC operation (streaming, self-updating, etc.) may be able to utilize the vulnerabilities in any or all of these cryptographic dependencies to undermine VLC's normal operation. In particular, CVE-2024-12243 could potentially be used to induce a denial-of-service within VLC, and CVE-2024-2236 could enable arbitrary decryption or signature forgery on any uses of RSA with PKCS#1v1.5 padding within VLC's networking APIs or modules. Recommendations The VideoLAN maintainers should update VLC's "contrib" build dependencies to use the latest versions of each of these dependencies. Longer term, the VideoLAN maintainers (consistent with TOB-VLC-6) should adopt a more structured and automatable approach to dependency version management, one that will enable automated detection and updating of vulnerable dependencies. References • Everlasting ROBOT: the Marvin Attack

HashEye 38 OSTIF - libVLC

PUBLIC Security Assessment

15. Weak cryptography in sftp module Type: Cryptography Finding ID: TOB-VLC-15 Target: modules/access/sftp.c

Description VLC provides an SFTP module built on top of libssh2. This module includes support for public key authentication, including with DSA keys: static const char defaultkeys[4][8] = { "rsa", "ed25519", "ecdsa", "dsa" }; Figure 15.1: Default allowed key types in VLC's SFTP module OpenSSH 7.0 and later, along with other major SSH implementations, disable DSA (via ssh-dss) due to a pervasive lack of support for keys larger than 1024 bits. libssh2 similarly disables DSA by default as of version 1.11.1. However, VLC is still built against version 1.11.0: LIBSSH2_VERSION := 1.11.0 LIBSSH2_URL := http://www.libssh2.org/download/libssh2-\$(LIBSSH2_VERSION).tar.gz Figure 15.2: Outdated version of libssh2 In addition to DSA support, VLC's current "contrib" build of libssh2 enables many unnecessary insecure defaults, including 3des-cbc (disabled in OpenSSH 7.4), blowfish-cbc (disabled in OpenSSH 7.6), and all RC4-based ciphers (disabled in OpenSSH 7.6). Exploit Scenario An attacker who can observe SFTP traffic between a victim VLC user and an SSH server may be able to mount a store-and-decrypt-later attack if the victim connects with ssh-dss. Similarly, per CVE-2016-2183, any SSH traffic encrypted with 3des-cbc may be susceptible to a Sweet32-style stack over long-duration connections. Recommendations Upgrade VLC to libssh2 version 1.11.1, which will disable ssh-dss by default. Additionally, VLC should include the following options in its "contrib" build of libssh2, as part of hardening the cryptographic suites offered during SFTP connections:

HashEye 39 OSTIF - libVLC

PUBLIC Security Assessment

- -DLIBSSH2_NO_MD5 to disable MD5-based HMACs and hashes, which OpenSSH disabled in 7.2 (2016-02) •
- DLIBSSH2_NO_3DES to disable 3des-cbc, which OpenSSH disabled in 7.4 (2016-12) • -
- DLIBSSH2_NO_HMAC_RIPEMD to disable RIPEMD-160 based HMACs, which OpenSSH disabled in 7.6 (2017-10)
- -DLIBSSH2_NO_BLOWFISH to disable blowfish-cbc, which OpenSSH disabled in 7.6 (2017-10) • -
- DLIBSSH2_NO_RC4 to disable RC4 based ciphers, which OpenSSH disabled in 7.6 (2017-10) • -
- DLIBSSH2_NO_CAST to disable cast128-cbc, which OpenSSH disabled in 7.6 (2017-10) • -
- DLIBSSH2_NO_MD5_PEM to disable support for old MD5-based encrypted private keys

HashEye 40 OSTIF - libVLC

PUBLIC Security Assessment

16. File-based keystore uses default file mode Type: Access Controls Finding ID: TOB-VLC-16 Target: modules/keystore/file.c

Description VLC provides secret store APIs via the keystore module. This module is backed by DPAPI on Windows and by the system keychain on macOS. On Linux the keystore module opportunistically uses KWallet if present, but falls back to a plaintext store on disk if KWallet is not available. When using the plaintext fallback, VLC calls `file_open`, which in turn calls `vlc_fopen`, which always passes `0666` to `vlc_open`: `int fd = vlc_open (filename, rwflags | oflags, 0666);` Figure 16.1: Call to `vlc_open` in `vlc_fopen` with `0666` mode This mode is then typically attenuated by the user's file creation mode mask (`umask`), i.e., typically `0022`. Consequently, the plaintext file created by the keystore module is typically created with mode `0644`, i.e., group-level and global read access. Consequently, all users (and groups) on the system can read any plaintext keystore created by VLC, regardless of the user running VLC. Exploit Scenario An attacker who obtains limited filesystem access (but not code execution or other extensive access) to a system running VLC may be able to exfiltrate the plaintext keystore, even if their filesystem access comes via an unrelated user or group. Recommendations The plaintext variant of the keystore module should be updated to prefer a default mode of `0600`, i.e., user read-write with no access for any other users or groups.

HashEye 41 OSTIF - libVLC

PUBLIC Security Assessment

17. Weak TLS defaults in GnuTLS module Type: Cryptography Finding ID: TOB-VLC-17 Target: modules/misc/gnutls.c

Description LibVLC uses GnuTLS to provide TLS APIs, including TLS support in the public libVLC networking APIs. GnuTLS provides cryptographic default configurations via "priority strings"; libVLC passes a user-controlled priority string (the `gnutls-priorities` variable) to `gnutls_priority_set_direct`. This user-controlled variable defaults to `NORMAL`, which enables various lower-security defaults. In particular: 1. TLS 1.0 and 1.1 are enabled. Both TLS 1.0 and 1.1 are susceptible to attacks that can't be mounted against newer versions of TLS, e.g., BEAST. Both TLS 1.0 and 1.1 are considered formally deprecated as of 2021, per RFC 8996, and have been disabled by default in macOS as of 2021 and Windows as of 2023. 2. The certificate verification profile permits selections with an equivalent symmetric security margin of 80 bits, i.e., equivalent to RSA 1024, DSA 1024, or a 160-bit ECC key. This is significantly below the standard security margin on the Web PKI of 128 bits, i.e., equivalent to RSA 2048 or a 256-bit ECC key.

`add_string ("gnutls-priorities", "NORMAL", PRIORITIES_TEXT, Figure 17.1: Default priorities string in VLC's gnutls module. Exploit Scenario An attacker who observes a TLS 1.0 connection between a LibVLC client and a server may be able to effect a BEAST-style attack. However, given that LibVLC is not a browser-style user-agent and does not have interactivity characteristics that are typically associated with the BEAST attack, we consider the difficulty of this attack high. Recommendations The gnutls priority string default in VLC should be changed from NORMAL to a custom priority string that disables TLS 1.0 and 1.1 while enabling a stronger set of certificate profile defaults. For example, a priority string of SECURE128:-VERS-ALL:+VERS-TLS1.2:+VERS-TLS1.3 would enable only the suites that offer 128 bits or more of security, and enforce TLS 1.2 or newer.`

HashEye 42 OSTIF - libVLC

PUBLIC Security Assessment

A remediating patch for this finding was opened with `videolan/vlc!7221` and was still awaiting review at report finalization time.

HashEye 43 OSTIF - libVLC

PUBLIC Security Assessment

18. GnuTLS module susceptible to Logjam Type: Cryptography Finding ID: TOB-VLC-18 Target: modules/misc/gnutls.c

Description Per TOB-VLC-17, libVLC's networking APIs make use of GnuTLS for TLS support. As part of configuring client TLS sessions, libVLC sets the minimum Diffie-Hellman prime size to 1024 bits:

```
static vlc_tls_t *gnutls_ClientSessionOpen(vlc_tls_client_t *crd, vlc_tls_t *sk, const char *hostname, const char *const *alpn) { vlc_tls_gnutls_t *priv = gnutls_SessionOpen(VLC_OBJECT(crd), GNUTLS_CLIENT, crd->sys, sk, alpn); if (priv == NULL) return NULL;
```

```
gnutls_session_t session = priv->session;
```

```
/* minimum DH prime bits */ gnutls_dh_set_prime_bits (session, 1024);
```

```
if (likely(hostname != NULL)) /* fill Server Name Indication */ gnutls_server_name_set (session, GNUTLS_NAME_DNS, hostname, strlen (hostname));
```

```
return &priv->tls; } Figure 18.1: Manual DH prime size configuration in TLS client session initialization
```

By doing this, libVLC allows servers to negotiate a connection with an unnecessarily small Diffie-Hellman group, in turn exposing connections to Logjam-style attacks. Exploit Scenario An attacker who observes a weak Diffie-Hellman exchange between a LibVLC client and an HTTPS server may be to effect a Logjam-style attack, thereby passively eavesdropping on the connection.

HashEye 44 OSTIF - libVLC

PUBLIC Security Assessment

Recommendations Remove the `gnutls_dh_set_prime_bits` call entirely, as GnuTLS will normally derive a proper Diffie-Hellman prime size from the global priority string settings. In particular, with the recommendations in TOB-VLC-17, GnuTLS should require a DH prime of at least 2048 bits, which is considered acceptable. If the call to `gnutls_dh_set_prime_bits` cannot be removed for whatever reason, it should be updated to explicitly require a prime size of at least 2048 bits. This finding was remediated with `videolan/vlc!7220`.

HashEye 45 OSTIF - libVLC

PUBLIC Security Assessment

19. Undocumented proxy server used for AcoustID lookups Type: Data Exposure Finding ID: TOB-VLC-19 Target: modules/misc/webservices/acoustid.c

Description VLC contains an AcoustID client, implemented via the `acoustid` module. This module contacts an AcoustID API server, sending an audio fingerprint along with an API key to the configured server. However, if a server is not configured, the `acoustid` module falls back to `ACOUSTID_ANON_SERVER` and `ACOUSTID_ANON_SERVER_PATH`, i.e., `fingerprint.videolan.org/acoustid.php`:

```
if( p_cfg->psz_server ) { if( unlikely(asprintf( &psz_url, "https://%s/v2/lookup" "?client=%s" "&meta=recordings+tracks+usermeta+releases" "&duration=%d" "&fingerprint=%s", p_cfg->psz_server, p_cfg->psz_apikey ? p_cfg->psz_apikey : "", p_data->i_duration, p_data->psz_fingerprint ) < 1 ) ) return VLC_EGENERIC; } else /* Use VideoLAN anonymized requests proxy */ { if( unlikely(asprintf( &psz_url, "https://" ACOUSTID_ANON_SERVER ACOUSTID_ANON_SERVER_PATH "?meta=recordings+tracks+usermeta+releases" "&duration=%d" "&fingerprint=%s", p_data->i_duration, p_data->psz_fingerprint ) < 1 ) ) return VLC_EGENERIC; } Figure 19.1: AcoustID request URL construction, including fallback. #define ACOUSTID_ANON_SERVER "fingerprint.videolan.org" #define ACOUSTID_ANON_SERVER_PATH "/acoustid.php" Figure 19.2: Hardcoded anonymizing server URL components.
```

HashEye 46 OSTIF - libVLC

PUBLIC Security Assessment

After conducting a review of online documentation, including informal sources like the VideoLAN forums, we conclude that the "VideoLAN anonymized requests proxy" is entirely undocumented in public user- and developer-facing resources. We additionally failed to locate the source code for `acoustid.php` anywhere on the VideoLAN GitLab server. Exploit Scenario This is an informational finding, and does not represent a security risk per se. However, because of its undocumented

nature, the VideoLAN anonymizing proxy may represent a valuable or interesting target to an attacker who seeks to obtain information on VLC users. In particular, an attacker who obtains access to the anonymizing proxy will gain access to fingerprints as well as device location information (in the form of IPs) that enables behavioral profiling of VLC users. Recommendations Conduct a review of the anonymizing proxy service for basic web service security practices. Additionally, the source code of the anonymizing proxy service be made available for external review, and the VideoLAN maintainers should conduct a review of all logging/event management configuration to ensure that user-originated metadata (such as media fingerprints) are not retained unnecessarily. Additionally, VLCs should include mentions of the anonymizing proxy service in user- or developer-facing documentation for either the acoustid module or VLC's fingerprinting functionality more broadly.

HashEye 47 OSTIF - libVLC

PUBLIC Security Assessment

20. VLC mirrors use plaintext rsync Type: Authentication Finding ID: TOB-VLC-20 Target: VLC Infrastructure

Description The VLC project coordinates a network of mirrors, which are maintained by third parties (including project sponsors). These provide load distribution when downloading official builds of VLC, as videolan.org will internally redirect official download requests to a local mirror. For example, when requesting the most recent build of VLC (3.0.21) from videolan.org on the US East Coast, <https://get.videolan.org/vlc/3.0.21/macosx/vlc-3.0.21-arm64.dmg> is redirected to <https://opencolo.mm.fcix.net/videolan-ftp/vlc/3.0.21/macosx/vlc-3.0.21-arm64.dmg>. Mirror operators have basic requirements, including serving mirrored resources over HTTPS. These requirements are documented on the Mirrors page of the VideoLAN Wiki. To initialize a mirror, operators are told to use anonymous rsync to obtain an initial mirror copy from the VideoLAN rsync daemon (signified by rsync:// as the protocol): `rsync --verbose --recursive --times --links \ --hard-links --perms --stats --delete-after \ --timeout=300 rsync://rsync.videolan.org/videolan-ftp \ /path/to/repository/destination` Figure 20.1: Formatted mirror initialization instructions from the VideoLAN Wiki Subsequent mirror updates are performed against the same rsync daemon on an hourly timer. Finally, mirror operators are told to enable their own anonymous read-only rsync daemon, with a similar configuration to that of the canonical VideoLAN host:

```
[videolan] path = /path/to/repository/destination comment = VideoLAN repository uid = nobody gid = nogroup read only = yes
```

 Figure 20.2: Read-only rsync configuration for mirror operators

HashEye 48 OSTIF - libVLC

PUBLIC Security Assessment

Mirrors are periodically checked for status and synchronization state by VideoLAN. VideoLAN employs Mirrorbits for this operation, which uses the same anonymous rsync daemon technique: if

```
!strings.HasPrefix(rsyncURL, "rsync://") { return 0, fmt.Errorf("%s does not start with rsync://", rsyncURL) }
```

```
u, err := url.Parse(rsyncURL) if err != nil { return 0, err }
```

```
// Extract the credentials if u.User != nil { if u.User.Username() != "" { env = append(env, fmt.Sprintf("USER=%s", u.User.Username())) } if password, ok := u.User.Password(); ok { env = append(env, fmt.Sprintf("RSYNC_PASSWORD=%s", password)) } }
```

```
// Remove the credentials from the URL as we pass them through the environment u.User = nil }
```

```
// Don't use the local timezone, use UTC env = append(env, "TZ=UTC")
```

```
cmd := exec.Command("rsync", "-r", "--no-motd", "--timeout=30", "--contimeout=30", "--exclude=~tmp~/", u.String())
```

 Figure 20.3: Anonymous rsync scanning within Mirrorbits Consequently, the entire mirror management lifecycle (initialization, periodic updates, health and timeliness checks from VideoLAN) is handled over anonymous rsync. When used in daemon mode, rsync performs connections on port 873 by default. These connections use the normal rsync protocol, but without a remote shell protocol (such as SSH) for transport. Consequently, connections performed using the rsync daemon are plain-text only and can be inspected or modified by an attacker with network-level access. Because the VLC mirroring scheme exclusively uses rsync in daemon mode, an attacker with network access has the ability to persistently deceive both VideoLAN and the mirror host about the

integrity of hosted files: 1. During mirror initialization or synchronization, the attacker can manipulate the contents of files as they please;

HashEye 49 OSTIF - libVLC

PUBLIC Security Assessment

2. During mirror status and consistency checks (via Mirrorbits), the attacker can submit valid responses back to VideoLAN while retaining manipulated contents on the mirror itself. Consequently, a VLC mirror can be made to persistently serve compromised assets in a manner that isn't detectable via the mirroring mechanism itself. Individual users may detect compromised assets by comparing them to the SHA-256 checksums published on the main VideoLAN website, but this requires non-technical users to perform a digest comparison that, in practice, the overwhelming majority of users likely fail to do. This has significant knock-on implications for TOB-VLC-1 and TOB-VLC-6, among others:

- The weak embedded public key documented in TOB-VLC-1 could be bypassed entirely by rewriting both source and build artifacts to contain an attacker-controlled public key.
- The HTTP-only downloads documented in TOB-VLC-6 are protected, in part, by SHA512SUMS files in the source distribution. However, an attacker who compromises the mirroring process can manipulate these at will. Because the VideoLAN website and download service automatically redirects to trusted mirrors, no further inducement is required to direct user traffic to an attacker-compromised mirror.

Exploit Scenario A victim who visits the official VideoLAN website to obtain an official build or source release of VLC may have their download redirected to a mirror that is inadvertently serving compromised assets due to an attacker with network-level access. This attacker can maintain their stealth even in the presence of mirror consistency checks by the VideoLAN upstream due to their network posture and the use of plain-text mirroring protocols. An attacker with the ability to mount this attack can do so pervasively by compounding with TOB-VLC-1, TOB-VLC-2, TOB-VLC-6, and so forth to ensure that all mirrored source builds are additionally compromised. Recommendations Short term, we strongly recommend that VideoLAN transition to an authenticated mirroring scheme. This mirroring scheme could be made maximally compatible with the existing scheme by retaining the use of rsync, but with read-only SSH for the transport and a well-known public key used for authentication by clients. Correspondingly, we strongly recommend that VideoLAN evaluate changes to Mirrorbits to support consistency checking via read-only SSH, along with an appropriate key-association or discovery mechanism for registered mirrors.

HashEye 50 OSTIF - libVLC

PUBLIC Security Assessment

Long term, we recommend that VideoLAN employ stronger integrity and honesty measures to prevent (intentional or unintentional) mirror compromise. Potential solutions here include:

- Minimizing third-party mirror use in favor of VideoLAN-controlled, CDN-style distribution.
- More aggressive integrity/honesty checks on mirrors beyond the Mirrorbits-mediated consistency checks, such as location- and time-randomized verification of assets directly from public HTTPS endpoints (rather than the rsync daemon).

HashEye 51 OSTIF - libVLC

PUBLIC Security Assessment

21. Undefined behavior for specific Musicbrainz API responses Type: Undefined Behavior Finding ID: TOB-VLC-21 Target: modules/misc/webseervices/musicbrainz.c

Description Two data checks for the response of the Musicbrainz API incorrectly access NULL pointers: if (media_array == NULL && media_array->size == 0) return false; Figure 21.1: NULL pointer access in musicbrainz.c:165 if (tracks == NULL && tracks->size == 0) return false; Figure 21.2: NULL pointer access in musicbrainz.c:173 An API response where either media_array or tracks is NULL will try to access the NULL pointer, which results in undefined behavior. Recommendations Replace the if conditions so that they use the logical OR operator (||) to avoid accessing the NULL pointer. This finding was remediated with videolan/vlc!7158.

HashEye 52 OSTIF - libVLC

PUBLIC Security Assessment

22. Address disclosure in ddummy decoder Type: Data Exposure Finding ID: TOB-VLC-22 Target: modules/codecdummy.c

Description VLC provides the ddummy module. This module offers “no-op” decoding of inputs, essentially synchronizing the VLC (or libVLC) application state against an input’s playback without actually launching a window. This module is documented under the “dummy modules” page on the VideoLan Wiki. The ddummy module offers a configurable boolean setting, dummy-save-es. When enabled, the ddummy module streams the raw codec data to a file on disk. The filename for this file is chosen by the ddummy module itself, and is constructed from stream.%p, where %p is the address of p_dec, i.e., the decoder_t being used by the module: static int OpenDecoderCommon(vlc_object_t *p_this, bool b_force_dump) { decoder_t *p_dec = (decoder_t*)p_this; char psz_file[10 + 3 * sizeof(p_dec)];

snprintf(psz_file, sizeof(psz_file), "stream.%p", (void *)p_dec); Figure 22.1: Construction of a filename containing an internal pointer. This effectively leaks the address of p_dec to the filesystem, giving an attacker with filesystem access information about the randomized layout of the program. An attacker may be able to chain this leakage with other information leakages (e.g., in logs and error messages) to perform an ASLR bypass. The ddummy module cannot be loaded and enabled without prior privileged access to VLC. Consequently, we consider this a purely informational finding. Recommendations Avoid logging or disclosing runtime addresses in unprivileged locations, such as filesystem paths or runtime logs. In this particular instance VLC should avoid using a runtime address as a de facto unique token, and should instead use an appropriate temporary filename construction. For external compatibility, VLC could produce file names that mirror the stream.%p form at but with the pointer component replaced by a random suffix.

HashEye 53 OSTIF - libVLC

PUBLIC Security Assessment

23. Pervasive address disclosure in logging components Type: Data Exposure Finding ID: TOB-VLC-23 Target: modules/logger/*

Description libVLC’s logging APIs and logging sinks are built around a common vlc_log_t type for logged messages, which is exposed in the public APIs as libvlc_log_t. This type includes two pointer-bearing members: i_object_id and tid. During log construction, i_object_id is initialized with the address of the internal vlc_logger used by the logging step, and tid is initialized with the return of the vlc_thread_id() helper: msg.i_object_id = (uintptr_t)(void *)loggerp; msg.psz_object_type = typename; msg.psz_module = module; msg.psz_header = NULL; msg.file = file; msg.line = line; msg.func = func; msg.tid = vlc_thread_id(); Figure 23.1: Initialization of vlc_log_t in vlc_valog vlc_thread_id(), in turn, is implemented in a platform-specific manner. On POSIX hosts without a discrete specialization, vlc_thread_id() is implemented by taking the address of a static thread_local member: VLC_WEAK unsigned long vlc_thread_id(void) { static thread_local unsigned char dummy;

static_assert (UINTPTR_MAX ≤ ULONG_MAX, "Type size mismatch"); return (uintptr_t)(void *)&dummy; } Figure 23.2: Implementation of vlc_thread_id() in src/posix/thread.c This implementation is marked as VLC_WEAK, as some POSIX hosts (like FreeBSD and Linux) have further specializations of vlc_thread_id() that use opaque, kernel-provided numeric thread IDs instead. However, macOS and other POSIX hosts without explicit specializations use the generic POSIX implementation.

HashEye 54 OSTIF - libVLC

PUBLIC Security Assessment

Consequently, there are up to two opaque “ID” addresses stored in each vlc_log_t: one that is uniformly present (i_object_id), and one that is conditionally present based on the host OS (tid). Both of these members are rendered as part of VLC’s various log sinks: fprintf(stream, ["GREEN"%0*"PRIxPTR GRAY"] ", ptr_width, meta→i_object_id); Figure 23.3: Disclosure of i_object_id in LogConsoleColor in modules/logger/console.c fprintf(stream, "[%0*"PRIxPTR]" ", ptr_width, meta→i_object_id); Figure 23.4: Disclosure of i_object_id in LogConsoleGray in modules/logger/console.c sd_journal_send("MESSAGE=%s", msg, "PRIORITY=%d", priorities[type], "CODE_FILE=%s", (meta→file ≠ NULL) ? meta→file : "", "CODE_LINE=%u", meta→line, "CODE_FUNC=%s", (meta→func ≠ NULL) ? meta→func : "", // "ERRNO=%d" "VLC_TID=%lu" /* change to OBJECT_TID if standardized */, meta→tid, "VLC_OBJECT_ID=%"PRIxPTR, meta→i_object_id, "VLC_OBJECT_TYPE=%s", meta→psz_object_type, "VLC_MODULE=%s", meta→psz_module, "VLC_HEADER=%s", (meta→psz_header ≠ NULL) ? meta→psz_header : "", NULL); Figure 23.5: Disclosure of both i_object_id and tid in Log in modules/logger/journal.c

```
if (asprintf(&formatz, "[%0*"PRIxPTR"/%lx] %s %s: %s", ptr_width, p_item→i_object_id, p_item→tid, p_item→psz_module, p_item→psz_object_type, format) < 0) return; Figure 23.6: Disclosure of both i_object_id and tid in AndroidPrintMsg in modules/logger/android.c
```

```
emscripten_log(prio, "[vlc.js: 0x*"PRIxPTR"/0x*"PRIxPTR"] %s %s: %s", p_item→i_object_id, p_item→tid, p_item→psz_module, p_item→psz_object_type, message); Figure 23.7: Disclosure of both i_object_id and tid in EmscriptenPrintMsg in modules/logger/emscripten.c
```

HashEye 55 OSTIF - libVLC

PUBLIC Security Assessment

These logging modules effectively leak up to two runtime addresses to potentially unprivileged logging sinks, disclosing information about the randomized layout of the program. An attacker may be able to chain these leakages with other information leakages to perform an ASLR bypass. Exploit Scenario An attacker with read access to the logging sinks described above may be able to obtain runtime addresses from a VLC or libVLC instance under a broad set of normal operating conditions, informing a subsequent ASLR bypass. Recommendations Avoid logging or disclosing runtime addresses in unprivileged locations, such as logging sinks. Where address logging is legitimately needed (e.g., for debugging), it should be gated by appropriate developer- or debugging-only build flags.

HashEye 56 OSTIF - libVLC

PUBLIC Security Assessment

24. Undefined behavior in SAT>IP port parsing Type: Undefined Behavior Finding ID: TOB-VLC-24 Target: modules/access/satip.c

Description A malformed port parameter in an RTSP Transport header within the satip module can induce undefined behavior within VLC. The satip module provides SAT>IP support. SAT>IP uses a variant of RTSP for its control plane, which the satip module implements via the rtsp_handle function. When rtsp_handle encounters a standard RTSP Transport header, it dispatches to parse_transport , which in turn parses each ; -delimited transport parameter. #define skip_whitespace(x) while(*x == ' ') x++ static enum rtsp_result rtsp_handle(stream_t *access, bool *interrupted) { access_sys_t *sys = access→p_sys; uint8_t buffer[512]; int rtsp_result = 0; bool have_header = false; size_t content_length = 0; size_t read = 0; char *in, *val;

```
/* Parse header */ while (!have_header) { in = net_readln_timeout(VLC_OBJECT(access), sys→tcp_socket, 5000, interrupted); if (in == NULL) break;
```

```
if (strncmp(in, "RTSP/1.0 ", 9) == 0) { rtsp_result = atoi(in + 9); } else if (strncmp(in, "Content-Base:", 13) == 0) { /* omitted for brevity */ } else if (strncmp("Transport:", in, 10) == 0) { val = in + 10; skip_whitespace(val);
```

```
if (parse_transport(access, val) != 0) { rtsp_result = VLC_EGENERIC; break; }
```

Figure 24.1: Header parsing loop in rtsp_handle

HashEye 57 OSTIF - libVLC

PUBLIC Security Assessment

```
static int parse_transport(stream_t *access, char *request_line) { access_sys_t *sys = access→p_sys; char *state; char *tok; int err;
```

```
tok = strtok_r(request_line, ";", &state); if (tok == NULL || strncmp(tok, "RTP/AVP", 7) != 0) return VLC_EGENERIC;
```

```
tok = strtok_r(NULL, ";", &state); if (tok == NULL || strncmp(tok, "multicast", 9) != 0) return 0;
```

```
while ((tok = strtok_r(NULL, ";", &state)) != NULL) { if (strncmp(tok, "destination=", 12) == 0) { memcpy(sys→udp_address, tok + 12, __MIN(strlen(tok + 12), UDP_ADDRESS_LEN - 1)); } else if (strncmp(tok, "port=", 5) == 0) { char port[6]; char *end;
```

```
memset(port, 0x00, 6); memcpy(port, tok + 5, __MIN(strlen(tok + 5), 5)); if ((end = strstr(port, "-")) != NULL) *end = '\0'; err = parse_port(port, &sys→udp_port); if (err) return err; } }
```

```
return 0; }
```

Figure 24.2: RTSP Transport header parameter parsing When the port header is encountered, its value is extracted and stored into sys→udp_port via parse_port, which uses atoi

```
for the string-to-integer conversion: static int parse_port(char *str, uint16_t *port) { int p = atoi(str); if (p < 0 || p > UINT16_MAX) return VLC_EINVAL;
```

```
*port = p;
```

```
return 0; }
```

Figure 24.3: Use of atoi in parse_port

HashEye 58 OSTIF - libVLC

PUBLIC Security Assessment

atoi's behavior is undefined when the converted value (i.e., the input string) is outside of the range of values representable within the int type. Since parse_transport does not validate that the contexts of the string are numeric (only 5 characters or fewer), this means that an attacker in control of a SAT>IP connection can induce undefined behavior within VLC by passing a port parameter such as port=wrong. Exploit Scenario An attacker in control of a SAT>IP connection (or just the RTSP control plane within the SAT>IP connection) can induce undefined behavior within VLC's satip module. Because the behavior is undefined, no concrete prediction about what happens can be made. However, a common behavior among libc implementations of atoi is to return 0 on conversion errors, meaning that the satip module attempts to open port 0 instead of a specific port. On Linux (and other POSIX-like hosts) this causes the kernel to dynamically select a suitable free port, which in turn may result in undesirable system resource consumption (e.g. port exhaustion over multiple attacker-controlled SAT>IP connections). Recommendations Avoid usage of atoi wherever possible. When string-to-integer conversion is needed, prefer APIs that can report errors, such as strtoul and strtoull.

HashEye 59 OSTIF - libVLC

PUBLIC Security Assessment

A. Vulnerability Categories The following tables describe the vulnerability categories used in this document. Vulnerability Categories Category Description Access Controls Insufficient authorization or assessment of rights Auditing and Logging Insufficient auditing of actions or logging of problems Authentication Improper identification of users Configuration Misconfigured servers, devices, or software components Cryptography A breach of system confidentiality or integrity Data Exposure Exposure of sensitive information Data Validation Improper reliance on the structure or values of data Denial of Service A system failure with an availability impact Error Reporting Insecure or insufficient reporting of error conditions Patching Use of an outdated software package or library Session Management Improper identification of authenticated users Testing Insufficient test methodology or test coverage Timing Race conditions or other order-of-operations flaws Undefined Behavior Undefined behavior triggered within the system

HashEye 60 OSTIF - libVLC

PUBLIC Security Assessment

B. Static Analysis Recommendations B.1. Semgrep Semgrep can be installed using pip by running python -m pip install semgrep. We recommend creating a free account and authenticating via semgrep login in order to get access to more analysis rules. To run Semgrep on the codebase, use the following command in the root directory of the project (running multiple predefined rules simultaneously by providing multiple --config arguments): semgrep --config auto To thoroughly understand the Semgrep tool refer to our HashEye Testing Handbook, where we aimed to streamline the Semgrep use and improve security testing effectiveness. Also, consider doing the following: • Limit results to error severity only by using the --severity ERROR flag. • Focus first on rules with high confidence and medium- or high-impact metadata. • Use the SARIF format (by using the --sarif Semgrep argument) with the SARIF Viewer for Visual Studio Code extension. This will make it easier to review the analysis results and drill down into specific issues to understand their impact and severity. B.2. clang-tidy clang-tidy can be added as a Makefile target and can be run as part of the CI/CD pipeline in order to run the static analysis checks before code is merged. Checks can be enabled/disabled individually, so in order to avoid having to fix all warnings at once before enabling clang-tidy in CI/CD, it can be enabled at first with all checks disabled, and then progressively add each check as the corresponding warnings are fixed. B.3. Docker image scanner The

VLC project maintains their own Docker image registry for the images used during CI/CD builds. A tool like grype can be used to monitor these images for vulnerabilities.

HashEye 61 OSTIF - libVLC

PUBLIC Security Assessment

C. Fuzzing Harness Recommendations As part of our review of libVLC, we conducted a review of libVLC's fuzz coverage and determined the following:

- VLC has in-tree fuzzer support, via the `test/vlc-demux-libfuzzer.c` harness.
- VLC is integrated into `oss-fuzz` for distributed fuzzing, per the `projects/vlc` harness. Via the `oss-fuzz` introspector, we determined that `oss-fuzz` currently contains 27 fuzzers: 26 for individual codecs and decoders (e.g., `vlc-demux-dec-libfuzzer-h264`), and a catch-all decoder fuzzer (`vlc-demux-dec-libfuzzer`). These fuzzers are runtime variants of the `vlc-demux-libfuzzer.c` harness above, which specializes its behavior based on its invoked name i.e. `argv[0]`. Based on that, we identified two areas for fuzzer harness improvement: additions via the existing polymorphic harness, and additions via new harnesses.

C.1. Demuxer and Decoder Harness Recommendations We recommend that the VideoLAN maintainers extend the `oss-fuzz` integration of the `vlc-demux-libfuzzer.c` harness to include the following additional demuxer and decoder targets:

- `aiff: modules/demux/aiff.c`
- `au: modules/demux/au.c`
- `caf: modules/demux/caf.c`
- `cdg: modules/demux/cdg.c`
- `dmxmus: modules/demux/dmxmus.c`
- `g64rtp: modules/demux/g64rtp.c`
- `gme: modules/demux/gme.c`
- `hx: modules/demux/hx.c`
- `mjpeg: modules/demux/mjpeg.c`
- `nsv: modules/demux/nsv.c`
- `nuv: modules/demux/nuv.c`
- `rawaud: modules/demux/rawaud.c`
- `dv: modules/demux/rawdv.c`
- `rawvid: modules/demux/rawvid.c`
- `sid: modules/demux/sid.cpp`
- `smf: modules/demux/smf.c`
- `tta: modules/demux/tta.c`
- `vc1: modules/demux/vc1.c`
- `vobsub: modules/demux/vobsub.c`

HashEye 62 OSTIF - libVLC

PUBLIC Security Assessment

Per Appendix D, adding these additional fuzzers to `oss-fuzz` should be straightforward, as each only requires a single (possibly empty) dictionary or seed file to trigger the appropriate fuzzer target construction in `oss-fuzz's` `build.sh`. Appendix D offers additional recommendations for ensuring that these new fuzzers (as well as existing demuxer/decoder fuzzers) provide coverage improvements.

C.2. New Harness Recommendations In addition to expanding use of the existing `vlc-demux-libfuzzer.c` harness, we recommend that the VideoLAN maintainers develop additional fuzzing harnesses for various high- and medium-impact API surfaces identified below. These API surfaces were identified through a combination of automatic and manual review of libVLC's APIs for parsing functionality and other functionalities that are amenable to fuzz testing. We group these API surfaces into two categories: "high-impact" and "medium-impact," corresponding to the relative prominence of the API and its attractiveness to an attacker (e.g. reachability from network sources). High-impact fuzzing harness candidates

- `src/misc/httpcookies.c`: The `cookie_parse` routine performs ad-hoc parsing on network-originated, attacker-controlled inputs (HTTP cookies).
- `src/text/url.c`: The `vlc_UrlParse` routine performs ad-hoc parsing on local and network-originated, attacker-controlled inputs (URLs, both from top-level user requests as well as indirect sources, e.g. sources with internal URL references).
- `src/network/http_auth.c`: The routines within this file perform ad-hoc parsing on network-originated, attacker-controlled inputs (HTTP authentication headers). In particular, the `vlc_http_auth_ParseWwwAuthenticateHeader` and `vlc_http_auth_ParseAuthenticationInfoHeader` routines perform ad-hoc string parsing.
- `src/network/httpd.c`: The routines within this file implement an HTTP daemon, including response handling. This handling occurs on network-originated, attacker-controlled inputs (HTTP response bodies). In particular, the `httpd_ClientRecv` routine performs extensive ad-hoc string parsing.
- `modules/access/ftp.c`: The `ftp` module implements an FTP client, which handles network-originated, attacker-controlled inputs (FTP server responses). In particular, the `ftp_RecvReply` routine performs ad-hoc string parsing both directly and indirectly via the callbacks it dispatches to.
- `modules/codecs/webvtt/css_parser.c`: The `webvtt` module contains an implementation of CSS, which is generated from lexer and parser definitions written in Flex and Bison, respectively. In particular, the `vlc_css_parser_ParseBytes` routine performs the lexing and parsing steps in a single call.

HashEye 63 OSTIF - libVLC

PUBLIC Security Assessment

Machine-generated lexers and parsers are typically less susceptible to memory corruption vulnerabilities. However, we recommend implementing a fuzzing harness for the CSS parser regardless, as it operates on network-originated, attacker-controlled inputs (WebVTT subtitles). Medium-impact fuzzing harness candidates

- `modules/access/satip.c`: The `satip` module contains multiple ad-hoc parsing routines that operate on network-originated, attacker-controlled inputs. In particular, the `rtsp_handle`, `parse_transport` and `parse_session` routines perform ad-hoc string parsing. Additionally, the `parse_port` routine appears to contain undefined behavior, per TOB-VLC-24.
- `modules/demux/json.c`: This file exposes a JSON parsing API, implemented on top of lexer and parser definitions written in Flex and Bison, respectively. This JSON parser is used throughout VLC, including in network-facing modules, to process attacker-controllable inputs. Like with the CSS parser in the `webvtt` module, this parser is expected to be less susceptible to memory corruption vulnerabilities than an equivalent ad-hoc parser would be. However, we still recommend implementing a fuzzing harness for it, given its pervasive status in the VLC codebase.
- `src/config/cmdline.c`: This file exposes libVLC's command-line parsing APIs. These APIs perform extensive ad-hoc parsing on locally-controlled inputs, i.e. command line arguments. In particular, the `config_LoadCmdLine` routine performs ad-hoc parsing prior to a more structured processing phase with `vlc_getopt_long`. Despite not being typically exposed over the network, command-line arguments are widely exposed to end users, including untrusted end users. This can lead to privilege escalation, e.g. if the program is a SUID binary as `vlc-wrapper` is. For additional information, see [cURL audit: How a joke led to significant findings](#).
- `modules/access/dcp/dcpdecrypt.cpp`: The `dcp` module contains DCP decryption routines that perform ad-hoc DER parsing, including ad-hoc parsing of DER tags and DER-encoded RSA keys. The same module additionally contains ad-hoc parsing of the PEM envelope format, as a preprocessing step for DER decoding. In particular, the `RSAPKey::readDER`, `RSAPKey::parseTag`, and `RSAPKey::readPEM` routines each perform ad-hoc string parsing, some of which appears to have been adapted from test-only driver code in `libgcrypt`. Per this appendix's recommendations, we developed three additional LibFuzzer-based Fuzzers for three internal APIs within libVLC: `vlc_UrlParse`, `vlc_css_parser_ParseBytes`, and `json_parse`. These parsers were delivered to the VideoLAN maintainers via a confidential issue on the VideoLAN GitLab.

HashEye 64 OSTIF - libVLC

PUBLIC Security Assessment

D. Fuzzer Corpus Recommendations In addition to our review of fuzzing harnesses in Appendix C, we conducted a review of libVLC's existing harness configurations. In particular, we conducted a review of existing fuzzing setups for adequate corpus configuration (in the form of seed inputs and fuzzing dictionaries). As part of that review, we determined the following:

- The `vlc-fuzz-corpus` repository is a central storage of fuzzing corpuses for VLC's fuzzers (currently just those produced by `vlc-demux-libfuzzer.c`).
- `vlc-fuzz-corpus` contains both dictionaries and seeds. The presence of a dictionary and/or seed with a given name (e.g. `mp4.dict`) influences the build and orchestration process within `oss-fuzz`, namely by ensuring that a corresponding fuzzer is generated (e.g. `vlc-demux-dec-libfuzzer-mp4`) from the top-level harness. Based on those determinations and a review of the current status of VLC in the `oss-fuzz` introspector, we make the following recommendations:

- Minimize individual corpus inputs, where possible. Several of the corpus inputs are nontrivially sized; for example, `pinball-cbr.wmv` in the `asf` corpus is nearly 800KB, and `ffv1_fuzz.avi` in the `avi` corpus is over 19MB. Large individual corpus members tend to produce slower and suboptimal fuzzing campaigns, as the fuzzing engine (LibFuzzer or AFL) needs to expend greater resources and explore a larger input space to perform meaningful mutations. In some cases, it may be possible to automatically minimize the pre-existing individual corpus inputs with `afl-cmin` or LibFuzzer's `-merge=1` mode. Additional information can be found in LibFuzzer's corpus documentation. For additional information on the performance characteristics of large corpus inputs, see AFL's performance tips.
- Increase overall corpus size and diversity, where possible. Many of the corpus seeds and dictionaries are stubs; for example, `dummy.dirac` in the `dirac` corpus is a 5-byte file containing the Dirac format magic bytes, as is `dummy.flac` in the `flac` corpus. Small and stubbed corpuses tend to produce suboptimal fuzzing campaigns, as the fuzzer has fewer coverage-informing inputs. Using the `flac` demuxer as an example: a high-level review of recent `oss-fuzz` introspector results suggests that corpus limitations are impeding coverage growth.

HashEye 65 OSTIF - libVLC

PUBLIC Security Assessment

In particular, significant portions of the flac demuxer remain uncovered due to insufficient discovery of e.g. metadata identifiers (seek tables, pictures, etc.). To improve overall corpus size and diversity, we make the following recommendations:

- Leverage Known Answer Test/test vector and conformance suites for seeding. For example, the IETF CELLAR Working Group maintains a flac-test-files repository that contains a variety of known-good and known-bad FLAC samples; these could be minimized and incorporated into vlc-fuzz-corpus. Similarly, the matroska-test-files repository could be used to improve the seeds for the mkv fuzzer (which is already relatively well-populated).
- Leverage LLMs for dictionary and seed generation. Large Language Models cannot replace traditional fuzzing mutation strategies for performance and feedback-sensitivity reasons. However, current families of LLMs are effective at one-off generation of AFL- and LibFuzzer-compatible dictionaries. Additionally, current-generation models are effective at higher-order generation, i.e. consuming a program's source and generating a script that produces high-coverage inputs for that program. For additional information, see FUZZING'24: "Is 'AI' useful for fuzzing?". Per this appendix's recommendations, we contributed additional libFuzzer-compatible dictionaries for the Dirac, FLAC, and MKV fuzzers. These dictionaries were contributed via VideoLAN.org/vlc-fuzz-corpus!3.

HashEye 66 OSTIF - libVLC

PUBLIC Security Assessment

E. Code Quality Recommendations This appendix contains findings and general recommendations that do not have immediate or obvious security implications, or were initiated but not fully investigated due to time constraints.

1. Pervasive use of raw C strings and ad-hoc C string parsing. VLC's codebase makes extensive use of conventional, null-terminated C strings ("raw" strings). These strings are parsed with conventional C standard library functions, such as strtok , strcspn , and strstr . "Raw" string parsing in C is a common source of spatial memory corruption, which in turn represents the plurality of vulnerabilities in in-the-wild exploits tracked by Google's Project Zero. As an alternative to "raw" parsing, we recommend that the VideoLAN maintainers investigate the integration of a higher-level and misuse-resistant string library, such as SDS or Better String.
2. Pervasive use of UB-prone C standard library functions. TOB-VLC-24 offers a single example of undefined behavior, originating from a call to atoi on non-numeric attacker-controlled input. However, a cursory search of libVLC and the module tree reveals dozens of other users of atoi and atof , along with similar C API functions with the same undefined behavior conditions. We recommend that the VideoLAN maintainers conduct a review of VLC's use of the following C standard library functions, and replace them as indicated:
 - atoi, atol, atoll: replace with strtol, strtoul, or strtoull with appropriate error checking.
 - atof: replace with strtod, strtold with appropriate error checking.
 - tolower and toupper: replace with vlc_ascii_tolower and vlc_ascii_toupper and perform domain validation as appropriate.More generally, we recommend that the VideoLAN maintainers use UBSan proactively as part of their development process to detect potential sources of undefined behavior.
3. Ad-hoc use of low-level cryptography APIs. Multiple findings (like TOB-VLC-4 and TOB-VLC-5) stem from direct use of low-level cryptographic APIs within VLC and libVLC, principally provided by gcrypt. Additionally, multiple components with VLC

HashEye 67 OSTIF - libVLC

PUBLIC Security Assessment

and libVLC perform implementations of cryptographic formats, including a partial implementation of the PGP packet format and a partial implementation of DER. We recommend that the VideoLAN maintainers conduct a thorough review of VLC's use of low-level cryptographic APIs, and evaluate higher-level replacements where possible. In particular, we make the following recommendations:

- Where digital signatures are needed, we recommend high-level APIs as provided by modern, misuse-resistant libraries like libsodium and Tink.
- Where nontrivial cryptographic container format parsing is needed (such as DER) we recommend that VLC use APIs from libtasn1 or another ASN.1 library, rather than separately implementing parsing routines. Separately, we strongly recommend that VLC eliminate unnecessary parsing of PGP packets, as part of a larger transition to modern, misuse-resistant digital signature APIs per above.
- Where encrypted channels (such as TLS and SSH) are needed, we recommend eliminating or minimizing low-level API calls (such as gnutls_dh_set_prime_bits) in favor of "blanket" high-level API calls that provide consistently secure defaults across all protocol subcomponents.

4. Ad-hoc dependency management practices. TOB-VLC-6, TOB-VLC-10, and TOB-VLC-14 reveal ad-hoc dependency management practices across various components of VLC and libVLC's build, test, and deployment lifecycles. These practices increase the complexity of component analysis within VLC and libVLC, in turn complicating determinations around

the outdatedness of dependencies or the presence of known vulnerabilities. Per the recommendations in the findings above as well as the analysis recommendations in Appendix B.3, we recommend that the VideoLAN maintainers investigate single-sourcing VLC's dependencies, and conducting automatic reviews of dependencies for both outdatedness and known vulnerabilities.

HashEye 68 OSTIF - libVLC

PUBLIC Security Assessment

About HashEye Founded in 2012 and headquartered in New York, HashEye provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel. We maintain an exhaustive list of publications at <https://github.com/hasheye-io/publications>, with links to papers, presentations, public audit reports, and podcast appearances. In recent years, HashEye consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon. We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries and government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom. HashEye also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash. To keep up to date with our latest news and announcements, please follow hasheye on X and explore our public repositories at <https://github.com/hasheye-io>. To engage us directly, visit our "Contact" page at <https://www.hasheye.io/contact> or email us at info@hasheye.io. HashEye, Inc. 228 Park Ave S #80688 New York, NY 10003 <https://www.hasheye.io> info@hasheye.io

HashEye 69 OSTIF - libVLC

PUBLIC Security Assessment

Notices and Remarks Copyright and Distribution © 2025 by HashEye, Inc. All rights reserved. HashEye hereby asserts its right to be identified as the creator of this report in the United Kingdom. HashEye considers this report public information; it is licensed to OSTIF under the terms of the project statement of work and has been made public at OSTIF's request. Material within this report may not be reproduced or distributed in part or in whole without HashEye's express written permission. The sole canonical source for HashEye publications is the HashEye Publications page. Reports accessed through sources other than that page may have been modified and should not be considered authentic. Test Coverage Disclaimer

HashEye performed all activities associated with this project in accordance with a statement of work and an agreed-upon project plan. Security assessment projects are time-boxed and often rely on information provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase. HashEye uses automated testing techniques to rapidly test software controls and security properties. These techniques augment our manual security review work, but each has its limitations. For example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. A project's time and resource constraints also limit their use.

HashEye 70 OSTIF - libVLC

PUBLIC Security Assessment