Binary Software Composition Analysis in Action: Finding vulnerable native libraries in Android Apps

Evgeny Zhukovsky

Founder DAP Solutions

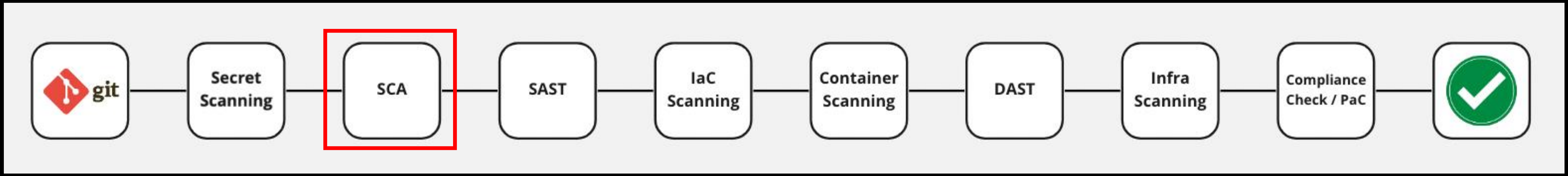
@spb_zhuk

Whoami

- Security Researcher
- 13+ years in CyberSecurity
- Founder of **DAP Solutions** (Pentest & AppSec Team)
- 10+ years in Security Teaching in University
- PhD, Associate Professor at the Peter the Great St.Petersburg Polytechnic University

Software composition analysis (SCA)

OWASP DevSecOps Guideline



Source: https://owasp.org/www-project-devsecops-guideline/

Software composition analysis (SCA) is an important step in SSDLC/DevSecOps practices that involves the automated process of identifying third-party components, such as open source software, in a codebase and tracking known vulnerabilities in them.

Limitations and Disadvantages of SCA

OFF ONE 2024

- Relies on information from package managers and build tools
- Uses only hashes to analyze binary dependencies
- Does not detect dependencies in static link binaries
- Does not detect v3rd party components manually added to the project on source code level

Binary Software Composition Analysis

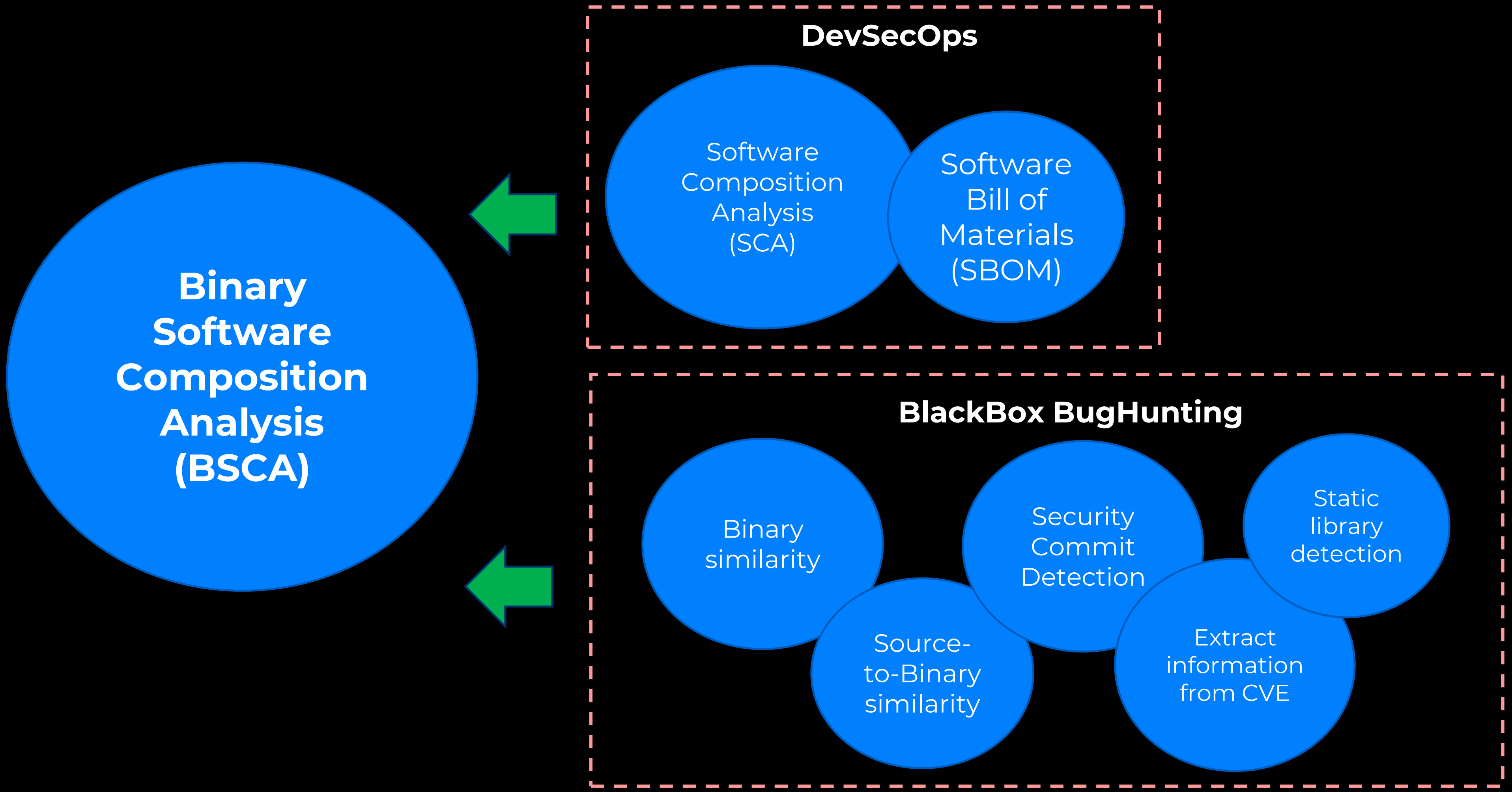
OFF ONE 2024

Binary Software Composition Analysis allows you to determine the composition of software based on the analysis of executable files

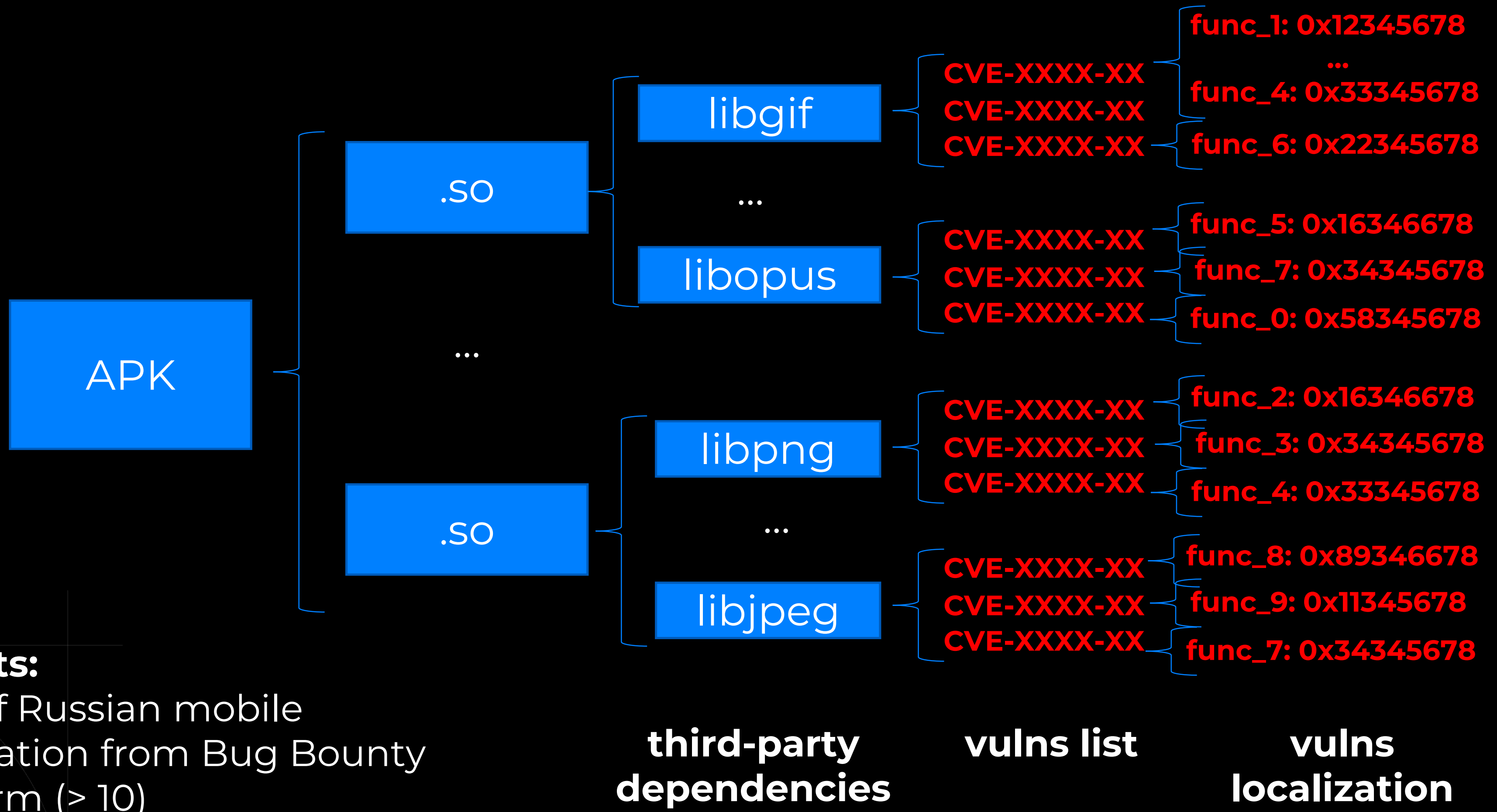
Advantages of BSCA:

- Detection of transitive dependencies and implicit dependencies (added manually)
- Does not require lengthy and complex setup and integration with development tools
- Does not require source code and can work based on analysis of binaries delivered to customers
- Can be used as a last mile check before delivery and detect vulnerabilities/backdoors introduced during the build process.
- Can be outsourced for external product analysis

BSCA domains of knowledge and challenges

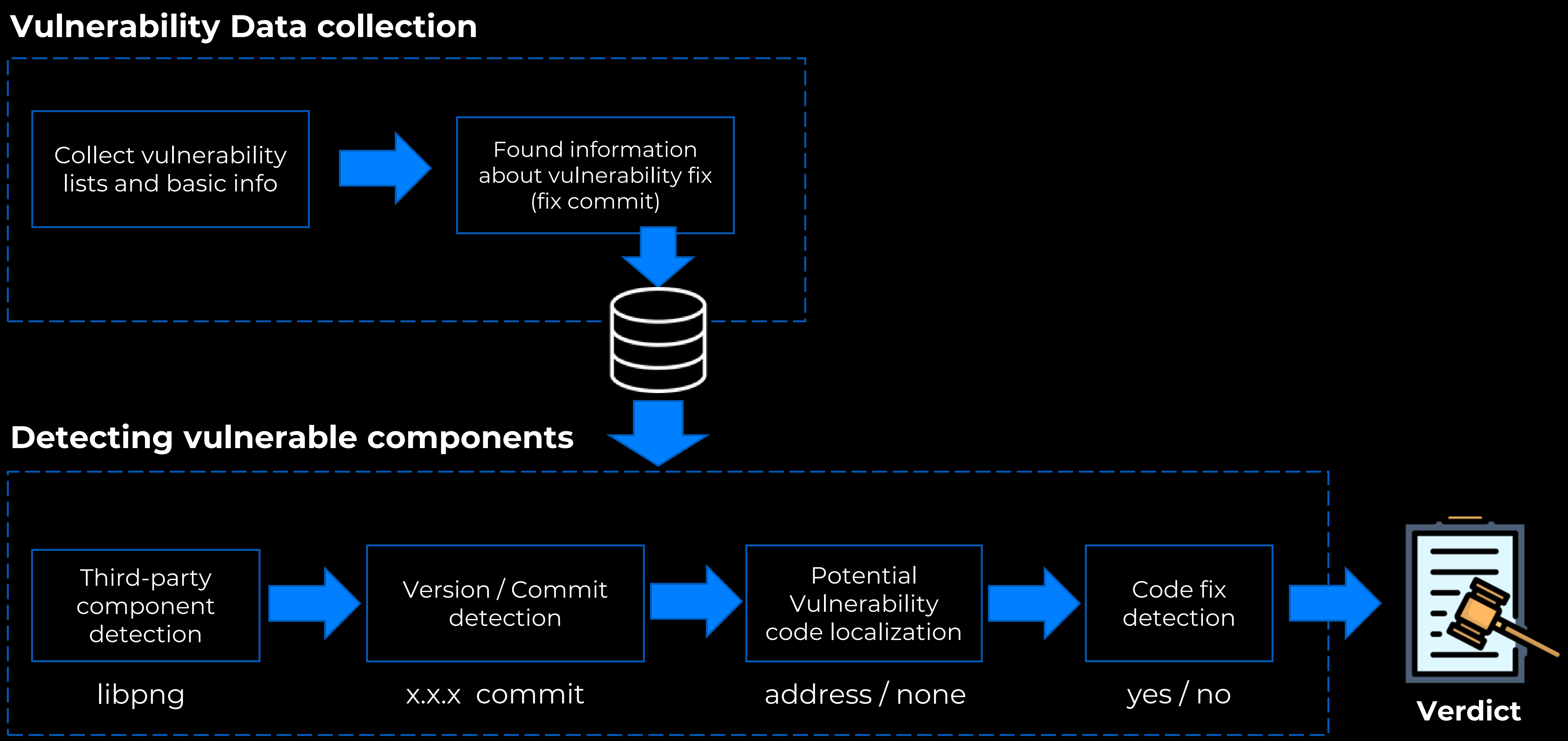


Use BSCA for Android Native libs

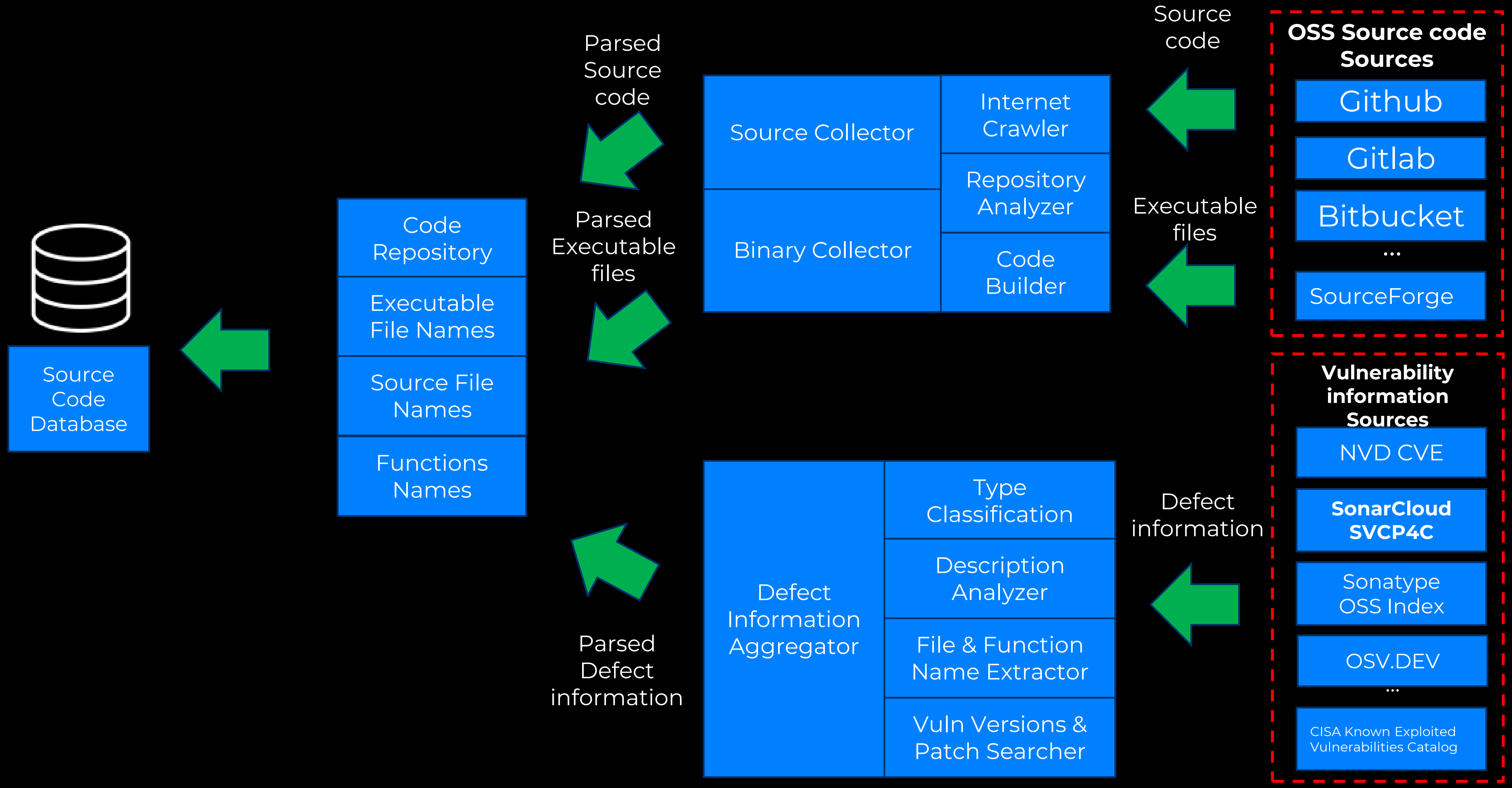


Targets:
APK of Russian mobile application from Bug Bounty platform (> 10)

BSCA phases


Vulnerability Data collection
Collect vulnerability lists and basic info
Found information about vulnerability fix (fix commit)
Detecting vulnerable components
Third-party component detection
libpng
Version / Commit detection
x.x.x commit
Potential Vulnerability code localization
address / none
Code fix detection
yes / no
Verdict

BSCA: Data collection



Search known vulnerabilities for libraries

- **CVE Databases**: not effective, only common info

...

- **Official Site:** vulns description, sometimes information about fix commits, and developer`s comments about ability real exploitation
- **Official repository** – fix commit info and description and sometimes PoCs
- **Bug Tracker System**: a full information about errors, sometimes PoCs or fuzzer`s testcase for reproduce, ASAN logs, stacktrace logs

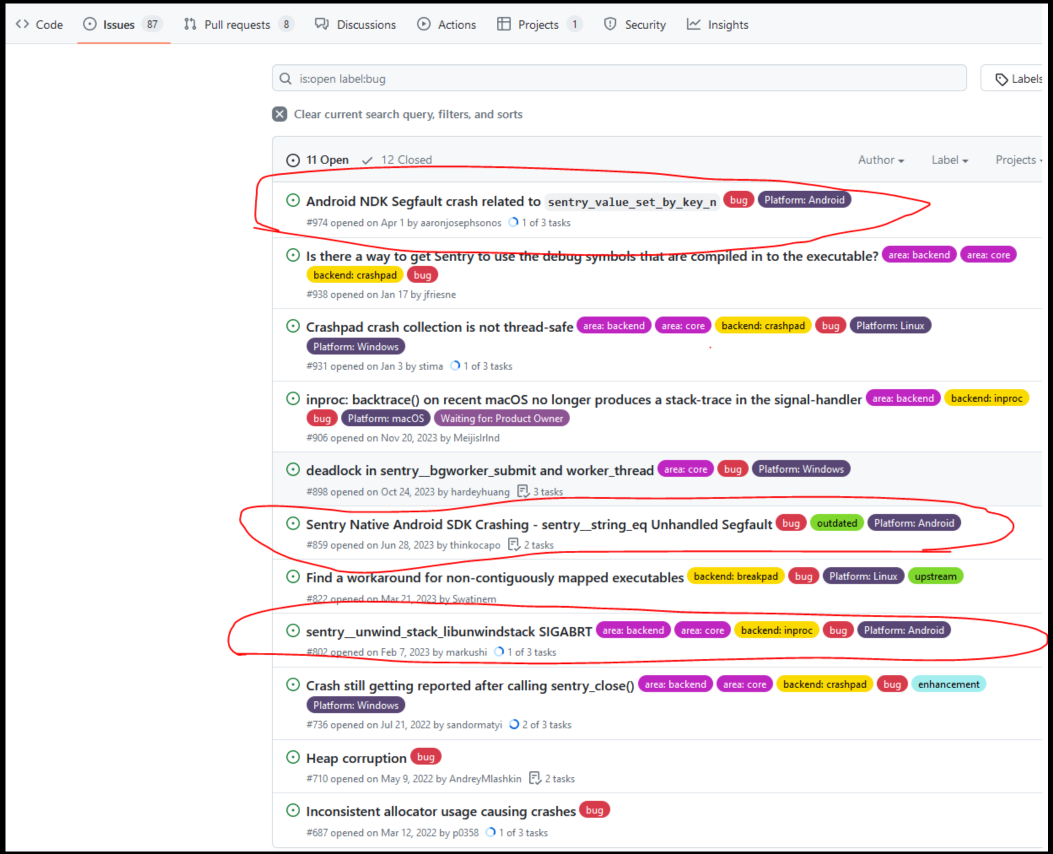
Useful sources of CVE info:

- https://vulnerabilityhistory.org
- https://osv.dev/
- https://ossindex.sonatype.org/
- https://fossa.com/
- https://github.com/dependabot
- https://snyk.io/

Search interesting errors in repository issues

Searching in **issues**:

- CVE
- vulnerability
- security
- critical, important
- exploit
- overflow: heap overflow, buffer overflow, integer overflow, overrun, override
- leak



CVE vs Silent Vulnerability Fix

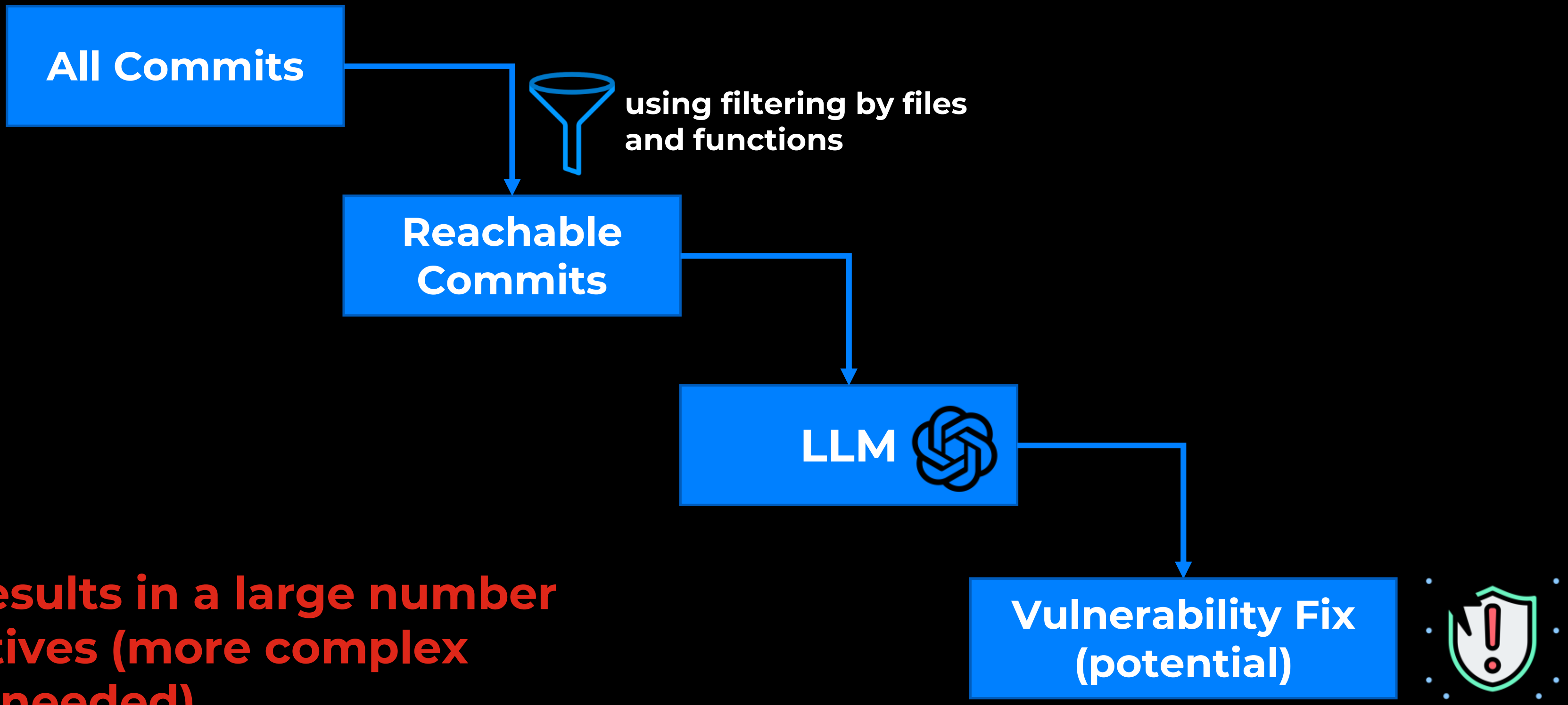
DevSecOps tracks CVEs, but why does fixing code without CVEs or issues not attract attention? Are you sure it is less dangerous?

Silent Vulnerability Fix relevance factors:

- all the attention on CVE
- CVEs are registered more on the initiative of external researchers
- most code fixes are made by developers, and they are not motivated to register CVEs and analyze the impact of changes on security for own code changes
- old commits are not covered by fuzzing
- there is no database of Silent Vulnerability Fixes

Searching Security Commits using LLM

Large Language Model helps automate analysis of potential security impact of patches

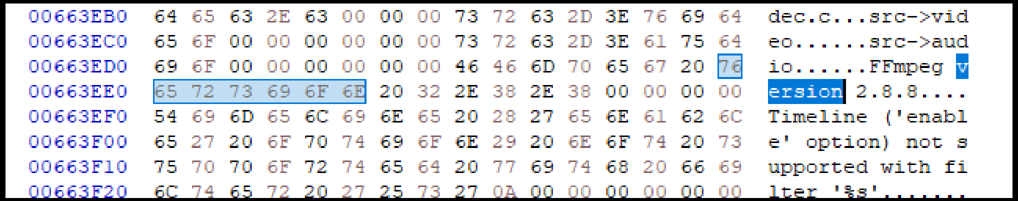


Using LLM results in a large number of false positives (more complex algorithm is needed)

Detecting Third-Party Components: Basics

Analyze **strings** and search name of components and version:

- can be name of component and version strings
- in Copyright
- in build artefacts (paths)
- in error / logs strings
- detection by name of functions from third-party library



List of **functions**:

- can detect version (know which functions in which versions were appear/disappear)

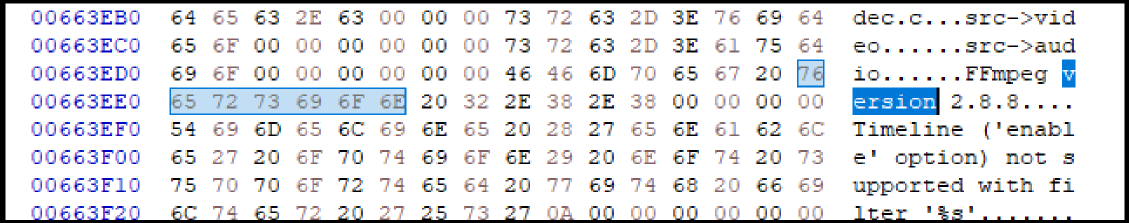
Detecting version based on string: bad idea

FF ONE 2024

Easy detection in this case: Binary of library contains the library name with version number

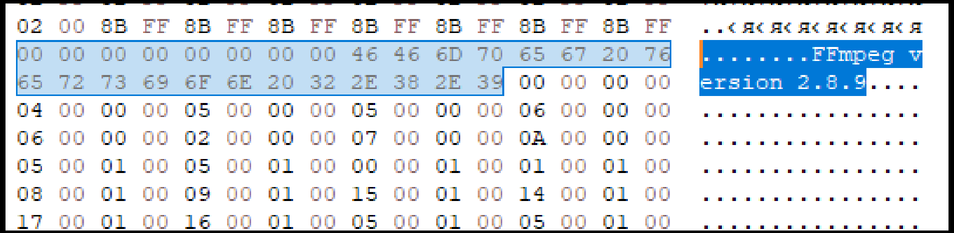
But is it correct?

FFmpeg version 2.8.8

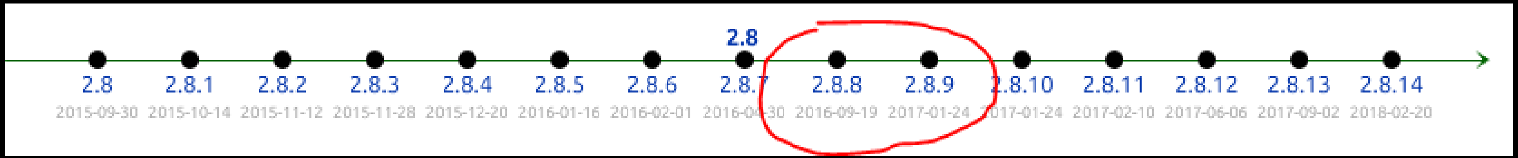


Ffmpeg 2.8.9 version

libavcodec-56



The Version 2.8.8 is dated 2016-09-19, but in reality it is more updated versions



Determining the presence of a vulnerability based on the version results in a large number of false positives

These versions can be affected by many vulnerabilities (from 2016 was register > 150 CVEs)

Approaches to detecting third-party components in binary code

OFF ONE 2024

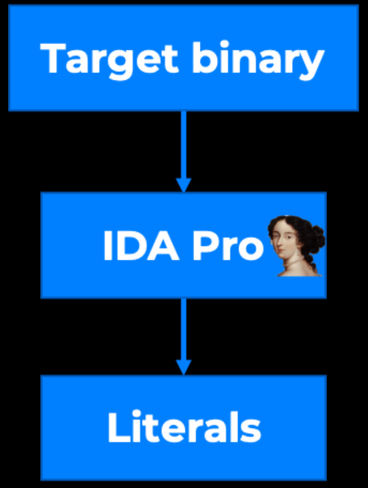
Popular approaches:

- Signatures
- Binary functions signatures
- Comparing code graph representation
- Assembly instruction's statistics
- Languages models based on decompiled / assembly code
- Machine learning on binaries
- Symbolic executions constraints for functions

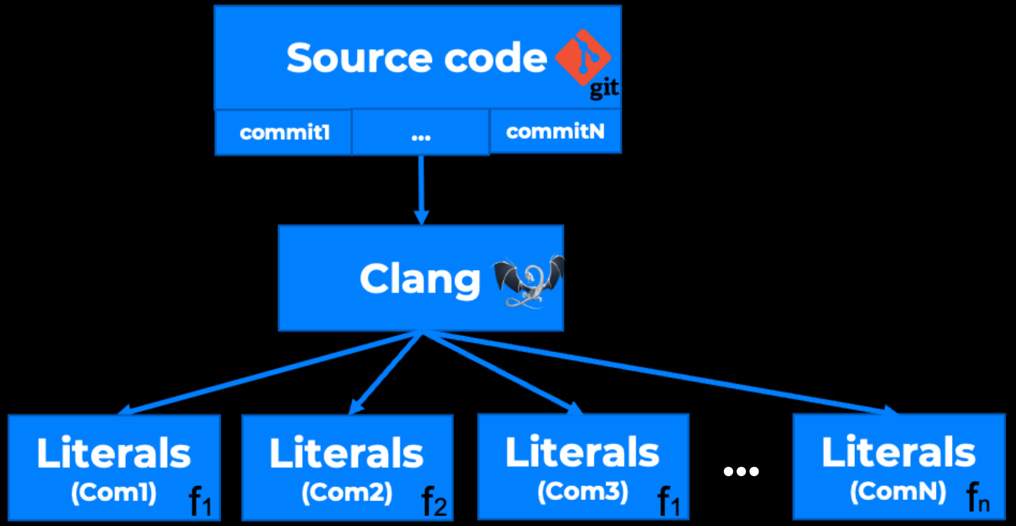
Tools:

- IDA Pro FLIRT, Lumina/ Ghidra FID
- Bindiff / Diaphora
- pigaious
- **BinaryAI Service**
- ...

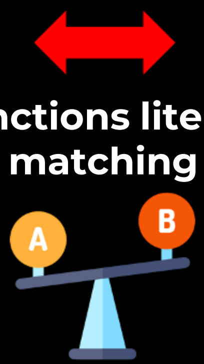
Functions and Commit detection based on literals matching



func_1: 4, ‘error input’, 0xFFFFEE, 255
func_2: ‘log(): function end’, 0xACFF, 2
...



func_1: C{com1, com3, ..., comN}: L{l1, l2, ..., lN}
func_2: C{com2, com5, ..., comN}: L{l3, l4, ..., lN}
...



Functions literals matching

Tasks:

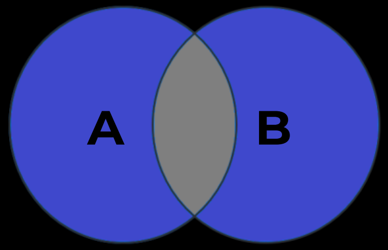
1. Functions localization
2. Commits detection

Function matching based on literals

Used combinations of the following metrics when comparing:

- literal uniqueness metric: $\sum_{l \in L} \frac{1}{\textbf{occurrence frequency (l)}}$

- Jaccard metric $K_J = \frac{\textbf{Intersection}\ (A, B)}{\textbf{Union}\ (A, B)}$

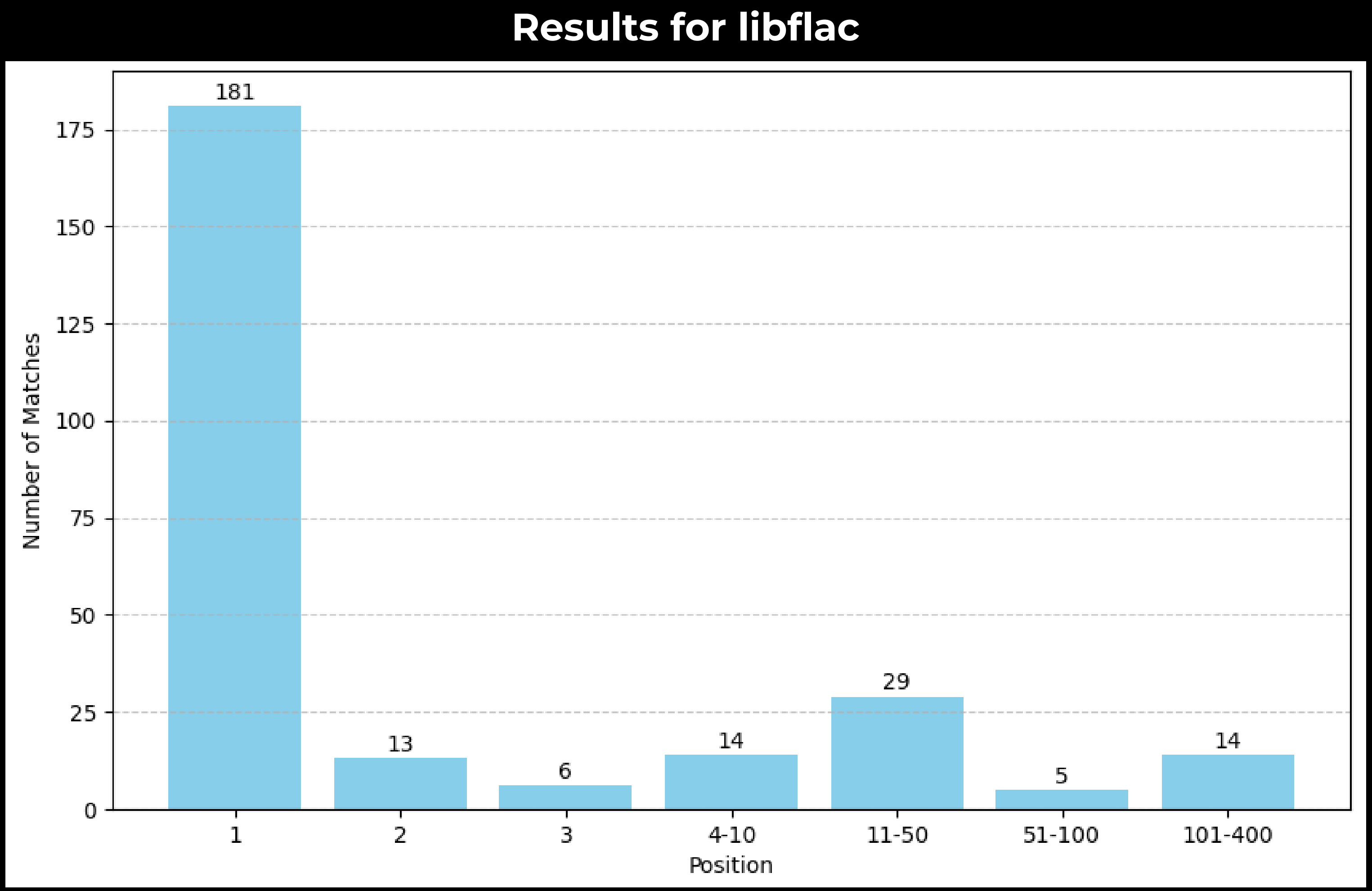


Function matching based on literals: Results

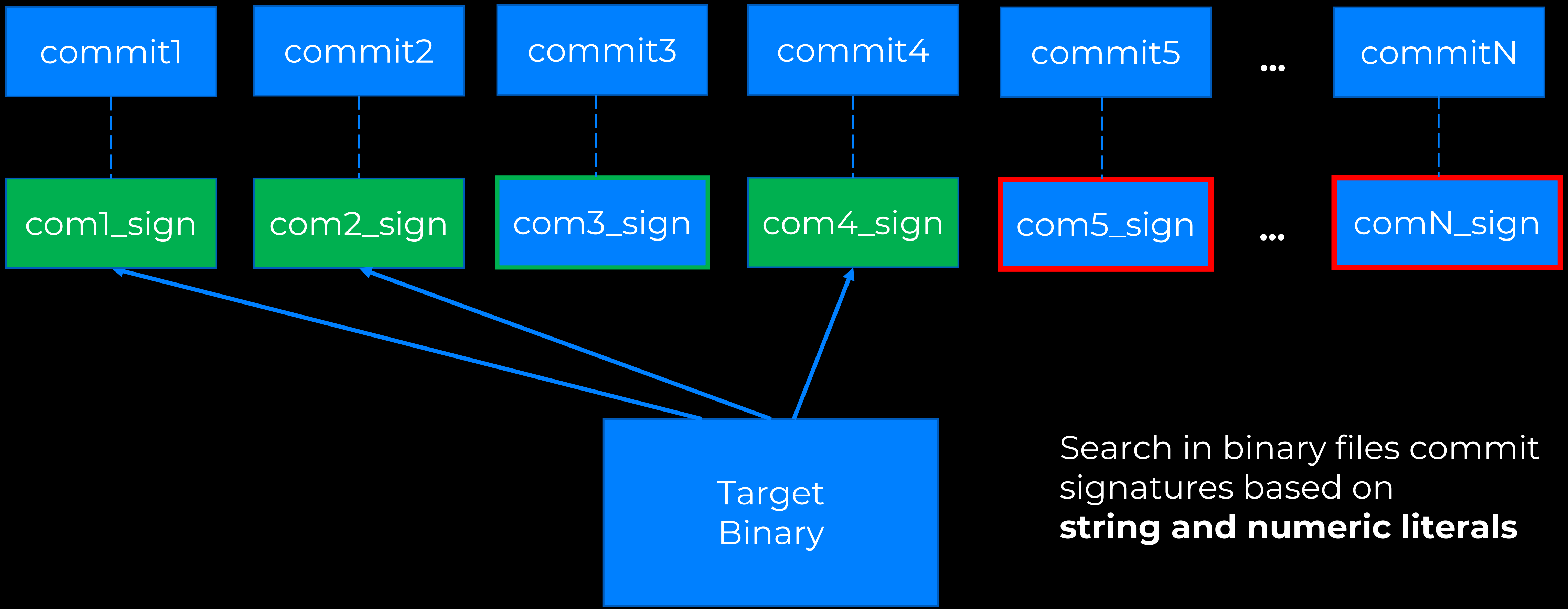
Taking file1 and search similar functions in file2 based on comparing literals

Sort functions from file2 and analyzing position of equal function in ranking

Results shows that the most functions were matched correctly based on comparing literals



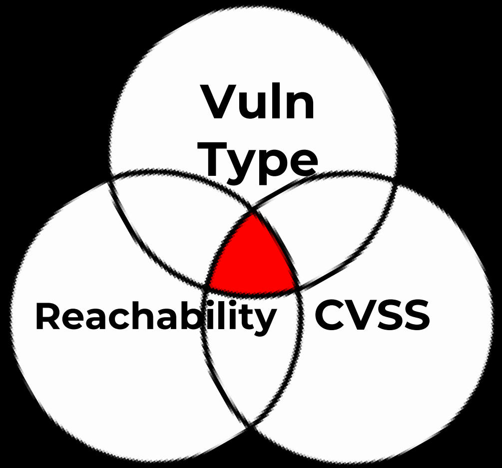
Version & commit detection



Search in binary files commit signatures based on **string and numeric literals**

Results of analysis using BSCA for native Libs

> 50 CVE in 8 OSS components
(most of them are in the ffmpeg)



~30 CVE for checking → **Manual Check**

Results of BSCA for FFmpeg libraries

Found several APK that use FFmpeg with version strings: 2.8.8 and 2.8.9

Functions in which the following vulnerabilities are known were localized:

- CVE-2018-14394
- CVE-2020-22016
- CVE-2020-22037
- CVE-2022-48434
- ...

Based on the versions, there will be potentially ~150 CVEs exposed

Based on commits detection was founded ~30 CVEs

~10 CVEs for manual checking

Let's start manual analysis...

CVE-2018-12459: fixed ☹

Decompiled target code

```
  *v14 = bswap32((v13 >> (2 - v11)) | (438 << v11));
  *(_QWORD *)(v3 + 16) += 4LL;
}
v15 = v11 + 30;
LABEL_16:
v16 = *(_QWORD *)(a1 + 8120);
*(_DWORD *)(a1 + 816) = v13;
v17 = *(_QWORD *)(a1 + 616);
*(_DWORD *)(v3 + 4) = v15;
v18 = *(int *)(v17 + 136);
if ( v16 <= 0 )
  v19 = (v16 - v18 + 1) / v18;
else
  v19 = v16 / v18;
v20 = v16 - *(_DWORD *)(v17 + 136) * v19;
v21 = v19 - *(_DWORD *)(a1 + 8108);
if ( v21 < 0 )
{
  av_log(0LL, 0LL, (__int64)"Assertion %s failed at %s:%d\n", "time_incr >= 0", "libavcodec/mpeg4videoenc.c", 1114LL);
  abort();
}
if ( v21 > 3600 )
{
  av_log((__int64 *)v17, 16LL, (__int64)"time_incr %d too large\n", (unsigned int)v21);
  return 4294967274LL;
}
if ( v21 )
{
  v22 = v13;
  do
  {
    while ( v15 > 1 )
    {
      v13 = (2 * v22) | 1;
      --v21;
002C1E0C ff_mpeg4_encode_picture_header:433 (2C1E0C)
```

Found code fix

Fix commit code

```
 int ff_mpeg4_decode_picture_header(Mpeg4DecContext *ctx, GetBitContext *gb);
 void ff_mpeg4_encode_video_packet_header(MpegEncContext *s);
diff --git a/libavcodec/mpeg4videoenc.c b/libavcodec/mpeg4videoenc.c
index 15d8825..4c1bce8 100644
--- a/libavcodec/mpeg4videoenc.c
+++ b/libavcodec/mpeg4videoenc.c
@@ -1086,7 +1086,7 @@ static void mpeg4_encode_vol_header(MpegEncContext *s,
 }

 /* write mpeg4 VOP header */
-void ff_mpeg4_encode_picture_header(MpegEncContext *s, int picture_number)
+int ff_mpeg4_encode_picture_header(MpegEncContext *s, int picture_number)
 {
     int time_incr;
     int time_div, time_mod;
@@ -1112,6 +1112,12 @@ void ff_mpeg4_encode_picture_header(MpegEncContext *s, int
     time_mod  = FFUMOD(s->time, s->avctx->time_base.den);
     time_incr = time_div - s->last_time_base;
     av_assert0(time_incr >= 0);
+
+    // This limits the frame duration to max 1 hour
+    if (time_incr > 3600) {
+        av_log(s->avctx, AV_LOG_ERROR, "time_incr %d too large\n", time_incr);
+        return AVERROR(EINVAL);
+    }
     while (time_incr--)
         put_bits(&s->pb, 1, 1);

@@ -1137,6 +1143,8 @@ void ff_mpeg4_encode_picture_header(MpegEncContext *s, int
         put_bits(&s->pb, 3, s->f_code);  /* fcode_for */
     if (s->pict_type == AV_PICTURE_TYPE_B)
         put_bits(&s->pb, 3, s->b_code);  /* fcode_back */
+
+    return 0;
 }

 static av_cold void init_uni_dc_tab(void)
diff --git a/libavcodec/mpegvideo_enc.c b/libavcodec/mpegvideo_enc.c
index 89ab1c3..55550d5 100644
--- a/libavcodec/mpegvideo_enc.c
+++ b/libavcodec/mpegvideo_enc.c
```

CVE-2018-12460: functionality is missing ☹

Decompiled target code

```
  *(_QWORD *)(a1 + 24) = ff_j_rev_dct1;
  *(_DWORD *)(a1 + 112) = 0;
  break;
default:
  v5 = a2[221];
  if ( (unsigned int)(v5 - 9) > 1 )
  {
    if ( v5 == 12 )
    {
      *(_QWORD *)(a1 + 32) = ff_simple_idct_put_12;
      *(_QWORD *)(a1 + 40) = ff_simple_idct_add_12;
      *(_QWORD *)(a1 + 24) = ff_simple_idct_12;
      *(_DWORD *)(a1 + 112) = 0;
    }
    else if ( v4 == 1 )
    {
      *(_QWORD *)(a1 + 32) = ff_jref_idct_put;
      *(_QWORD *)(a1 + 40) = ff_jref_idct_add;
      *(_QWORD *)(a1 + 24) = ff_j_rev_dct;
      *(_DWORD *)(a1 + 112) = 1;
    }
    else
    {
      if ( v4 == 20 )
      {
        *(_QWORD *)(a1 + 32) = ff_faanidct_put;
        *(_QWORD *)(a1 + 40) = ff_faanidct_add;
        *(_QWORD *)(a1 + 24) = ff_faanidct;
      }
      else
      {
        *(_QWORD *)(a1 + 32) = ff_simple_idct_put_8;
        *(_QWORD *)(a1 + 40) = ff_simple_idct_add_8;
        *(_QWORD *)(a1 + 24) = ff_simple_idct_8;
      }
      *(_DWORD *)(a1 + 112) = 0;
    }
  }
  else
  {
    *(_QWORD *)(a1 + 32) = ff_simple_idct_put_10;
    *(_QWORD *)(a1 + 40) = ff_simple_idct_add_10;
    *(_QWORD *)(a1 + 24) = ff_simple_idct_10;
    *(_DWORD *)(a1 + 112) = 0;
  }
  break;
}
*(_QWORD *)(a1 + 8) = sub_287400;
*(_QWORD *)a1 = sub_2872E8;
000818BC ff_idctdsp_init:59 (818BC)
```

Missing codec mode code that is vulnerable

Fix commit code

```
        /* 10-bit MPEG-4 Simple Studio Profile requires a higher
precision IDCT
           However, it only uses idct_put */
-       if (avctx->codec_id == AV_CODEC_ID_MPEG4 && avctx->profile ==
FF_PROFILE_MPEG4_SIMPLE_STUDIO)
+       if (c->mpeg4_studio_profile)
            c->idct_put              = ff_simple_idct_put_int32_10bit;
        else {
            c->idct_put              = ff_simple_idct_put_int16_10bit;
            c->idct_add              = ff_simple_idct_add_int16_10bit;
            c->idct                  = ff_simple_idct_int16_10bit;
        }
        c->perm_type             = FF_IDCT_PERM_NONE;
    } else if (avctx->bits_per_raw_sample == 12) {
        c->idct_put              = ff_simple_idct_put_int16_12bit;
        c->idct_add              = ff_simple_idct_add_int16_12bit;
        c->idct                  = ff_simple_idct_int16_12bit;
        c->perm_type             = FF_IDCT_PERM_NONE;
    } else {
        if (avctx->idct_algo == FF_IDCT_INT) {
            c->idct_put  = ff_jref_idct_put;
            c->idct_add  = ff_jref_idct_add;
            c->idct      = ff_j_rev_dct;
            c->perm_type = FF_IDCT_PERM_LIBMPEG2;
#if CONFIG_FAANIDCT
        } else if (avctx->idct_algo == FF_IDCT_FAAN) {
            c->idct_put  = ff_faanidct_put;
            c->idct_add  = ff_faanidct_add;
            c->idct      = ff_faanidct;
            c->perm_type = FF_IDCT_PERM_NONE;
#endif /* CONFIG_FAANIDCT */
        } else { // accurate/default
            /* Be sure FF_IDCT_NONE will select this one, since it uses
FF_IDCT_PERM_NONE */
            c->idct_put  = ff_simple_idct_put_int16_8bit;
            c->idct_add  = ff_simple_idct_add_int16_8bit;
            c->idct      = ff_simple_idct_int16_8bit;
            c->perm_type = FF_IDCT_PERM_NONE;
```

CVE-2018-14394: vulnerable ☺

Decompiled target code

```
if ( v27 == 73728 )
{
  if ( n[0] <= 0 )
  {
    LODWORD(samples_in_chunk) = 0;
    if ( *(_DWORD *)(trk + 108) )
      goto LABEL_34;
  }
  else
  {
    v33 = *(unsigned __int16 **)(pkt_AVPacket + 24);
    v72 = 0;
    LODWORD(samples_in_chunk) = 0;
    do
    {
      LODWORD(samples_in_chunk) = samples_in_chunk + 1;
      v72 += word_70E20[(*((_BYTE *)v33 + v72) >> 3) & 0xF];
    }
    while ( v72 < n[0] && (unsigned int)samples_in_chunk <= 0x63 );
    if ( (unsigned int)samples_in_chunk > 1 )
    {
      av_log(s_AVFormatContext, 16LL, "fatal error, input is not a s
      return;
    }
    if ( *(_DWORD *)(trk + 108) )
      goto LABEL_37;
  }
}
else
{
  if ( v27 == 69633 || v27 == 69638 )
  {
    LODWORD(samples_in_chunk) = *(_DWORD *)(v11 + 476);
  }
  else
  {
    trk_samples_size = *(_QWORD *)(trk + 48);
    LODWORD(samples_in_chunk) = 1;
    if ( trk_samples_size )
      samples_in_chunk = n[0] / trk_samples_size;
  }
  if ( *(_DWORD *)(trk + 108) )
    goto LABEL_33;
}
v30 = *(_DWORD *)(v11 + 128);
if ( v30 > 0 )
{
```

AV_CODEC_ID_ADPCM_IMA_WAV = 69633;
AV_CODEC_ID_AMR_NB = 73728;

Fix commit code

```
    if (par->codec_id == AV_CODEC_ID_AMR_NB) {   73728
        /* We must find out how many AMR blocks there are in one packet */
        static const uint16_t packed_size[16] =
            {13, 14, 16, 18, 20, 21, 27, 32, 6, 0, 0, 0, 0, 0, 0, 1};
        int len = 0;

        while (len < size && samples_in_chunk < 100) {
            len += packed_size[(pkt->data[len] >> 3) & 0x0F];
            samples_in_chunk++;
        }
        if (samples_in_chunk > 1) {
            av_log(s, AV_LOG_ERROR, "fatal error, input is not a single packet, impl
            return -1;
        }
    } else if (par->codec_id == AV_CODEC_ID_ADPCM_MS ||
               par->codec_id == AV_CODEC_ID_ADPCM_IMA_WAV) {   69633
        samples_in_chunk = trk->par->frame_size;
    } else if (trk->sample_size)
        samples_in_chunk = size / trk->sample_size;
    else
        samples_in_chunk = 1;

+   if (samples_in_chunk < 1) {
+       av_log(s, AV_LOG_ERROR, "fatal error, input packet contains no samples\n");
+       return AVERROR_PATCHWELCOME;
+   }
+
    /* copy extradata if it exists */
    if (trk->vos_len == 0 && par->extradata_size > 0 &&
        !TAG_IS_AVCI(trk->tag) &&
```

check is missing

CVE-2020-22016: BoF - vulnerable ☺

FF ONE 2024

Decompiled target code

```
else
{
  if ( codec_id_v29 == 69633 || codec_id_v29 == 69638 )
  {
    LODWORD(samples_in_chunk_v31) = *(_DWORD *)(v11 + 476);
  }
  else
  {
    v30 = *(_QWORD *)(v9 + 48);
    LODWORD(samples_in_chunk_v31) = 1;
    if ( v30 )
      samples_in_chunk_v31 = n[0] / v30;
  }
  if ( *(_DWORD *)(v9 + 108) )
    goto LABEL_33;
}
v32 = *(_DWORD *)(v11 + 128);
if ( v32 > 0 )
{
  v33 = *(_DWORD *)(v9 + 84);
  if ( (v33 & 0xFEFBFFFF) != 842099041
    && (v33 & 0xFEFBFFFF) != 1882286433
    && codec_id_v29 != 100
    && v33 != 1852397121
    && v33 != 2021026145
    && (((v33 & 0xFFFBFFFF) - 892430689) & 0xFEFFFFFF) != 0 )
  {
    *(_DWORD *)(v9 + 108) = v32;
    v34 = av_malloc(v32);
    *(_QWORD *)(v9 + 112) = v34;
    if ( !v34 )
    {
LABEL_43:
      v40 = -12;
      goto LABEL_44;
    }
    memcpy(v34, *(const void **)(v11 + 120), *(int *)(v9 + 108));
    codec_id_v29 = *(_DWORD *)(v11 + 56);
  }
}
```

error in memory allocation

memset is missing

Fix commit code

```
            return -1;
        }
    } else if (par->codec_id == AV_CODEC_ID_ADPCM_MS ||
               par->codec_id == AV_CODEC_ID_ADPCM_IMA_WAV) {
        samples_in_chunk = trk->par->frame_size;
    } else if (trk->sample_size)
        samples_in_chunk = size / trk->sample_size;
    else
        samples_in_chunk = 1;

    if (samples_in_chunk < 1) {
        av_log(s, AV_LOG_ERROR, "fatal error, input packet contains no samples\n");
        return AVERROR_PATCHWELCOME;
    }

    /* copy extradata if it exists */
    if (trk->vos_len == 0 && par->extradata_size > 0 &&
        !TAG_IS_AVCI(trk->tag) &&
        (par->codec_id != AV_CODEC_ID_DNXHD)) {
        trk->vos_len  = par->extradata_size;
-       trk->vos_data = av_malloc(trk->vos_len);
+       trk->vos_data = av_malloc(trk->vos_len + AV_INPUT_BUFFER_PADDING_SIZE);
        if (!trk->vos_data) {
            ret = AVERROR(ENOMEM);
            goto err;
        }
        memcpy(trk->vos_data, par->extradata, trk->vos_len);
+       memset(trk->vos_data + trk->vos_len, 0, AV_INPUT_BUFFER_PADDING_SIZE);
    }
```

Checking the reachability of vulnerable code from APK

- Use Android Emulator (ARM) for running and debugging application
- Use Frida for analysis loaded native libraries and functions tracing
- Analyzing decompiled code of APK with JADX and JEB

- It is difficult for full automatization
- You need to understand app logic and its use cases
  to trigger vulnerable code

libFLAC: CVE-2020-0499 – vulnerable ☺

Decompiled target code

```
LABEL_48:
        result = FLAC__bitreader_read_raw_uint32(br_buffer, &v35, a4 - v24);
        if ( !(_DWORD)result )
          return result;
        v35 |= v25;
        *v4++ = -(((v14 << a4) | v35) & 1) ^ (((v14 << a4) | v35) >> 1);
        v10 = *br_buffer;
        cwords__br_consumed_words = *((unsigned int *)br_buffer + 5);
        br_consumed_bits = *((_DWORD *)br_buffer + 6);
        words = *((_DWORD *)br_buffer + 3);
        b = *(_DWORD *)(*br_buffer + 4 * cwords__br_consumed_words) << br_consumed_bits;
        if ( v4 < v6 )
        {
          v14 = 0;
          if ( (unsigned int)cwords__br_consumed_words >= words )
            continue;
        }
        goto LABEL_9;
      }
    }
```

check is missing

Fix commit code

```
incomplete_lsbs:
                                br->consumed_bits = 0;
                                br->consumed_words = cwords;
                        }

                        /* read the binary LSBs */
                        if(!FLAC__bitreader_read_raw_uint32(br, &lsbs, parameter - ucbits))
                                return false;
                        lsbs = x | lsbs;

                        /* compose the value */
                        x = (msbs << parameter) | lsbs;
                        *val++ = (int)(x >> 1) ^ -(int)(x & 1);
                        x = 0;

                        cwords = br->consumed_words;
                        words = br->words;
                        ucbits = FLAC__BITS_PER_WORD - br->consumed_bits;
-                       b = br->buffer[cwords] << br->consumed_bits;
+                       b = cwords < br->capacity ? br->buffer[cwords] << br->consumed_bits : 0;
                } while(cwords >= words && val < end);
        }

        if(ucbits == 0 && cwords < words) {
                /* don't leave the head word with no unconsumed bits */
```

libFLAC: CVE-2021-0561 – vulnerable ☺

FF ONE 2024

Decompiled target code

```c
v34[1] = *(_QWORD *)(_ReadStatusReg(ARM64_SYSREG(3, 3, 13, 0, 2)) + 40);
result = FLAC__bitwriter_get_buffer(*((_QWORD *)a1[1] + 996), &v33, &v32);
v5 = *a1;
if ( !(_DWORD)result )
{
  v8 = 8;
LABEL_8:
  *v5 = v8;
  return result;
}
if ( v5[1] )
{
  v6 = a1[1];
  *((_QWORD *)v6 + 1113) = v33;
  v7 = v6[2206];
  v6[2229] = v32;
  if ( v7 )
  {
    if ( !(unsigned int)FLAC__stream_decoder_process_single(*((_DWORD ***)v6 + 1102)) )
    {
      FLAC__bitwriter_release_buffer(*((_QWORD *)a1[1] + 996));
      FLAC__bitwriter_clear(*((_QWORD *)a1[1] + 996));
      v5 = *a1;
      if ( **a1 == 4 )
        return 0LL;
      v8 = 3;
      result = 0LL;
      goto LABEL_8;
    }
  }
  else
  {
    v6[2207] = 1;
```

check is missing

Fix commit code

```c
FLAC__ASSERT(FLAC__bitwriter_is_byte_aligned(encoder->private_->frame));

if(!FLAC__bitwriter_get_buffer(encoder->private_->frame, &buffer, &bytes)) {
        encoder->protected_->state = FLAC__STREAM_ENCODER_MEMORY_ALLOCATION_ERROR;
        return false;
}

if(encoder->protected_->verify) {
        encoder->private_->verify.output.data = buffer;
        encoder->private_->verify.output.bytes = bytes;
        if(encoder->private_->verify.state_hint == ENCODER_IN_MAGIC) {
                encoder->private_->verify.needs_magic_hack = true;
        }
        else {
                if(!FLAC__stream_decoder_process_single(encoder->private_->verify.decoder)) {
                if(!FLAC__stream_decoder_process_single(encoder->private_->verify.decoder)
                    || (!is_last_block
                        && (FLAC__stream_encoder_get_verify_decoder_state(encoder) == FLAC__STREAM_DECODER_END_OF_STREAM))) {
                        FLAC__bitwriter_release_buffer(encoder->private_->frame);
                        FLAC__bitwriter_clear(encoder->private_->frame);
                        if(encoder->protected_->state != FLAC__STREAM_ENCODER_VERIFY_MISMATCH_IN_AUDIO_DATA)
                                encoder->protected_->state = FLAC__STREAM_ENCODER_VERIFY_DECODER_ERROR;
                        return false;
                }
        }
}
```

What was found after manual check

Only half of all potential CVE found have been manually checked so far.

Vulnerable code from the following native libraries was found to be used:

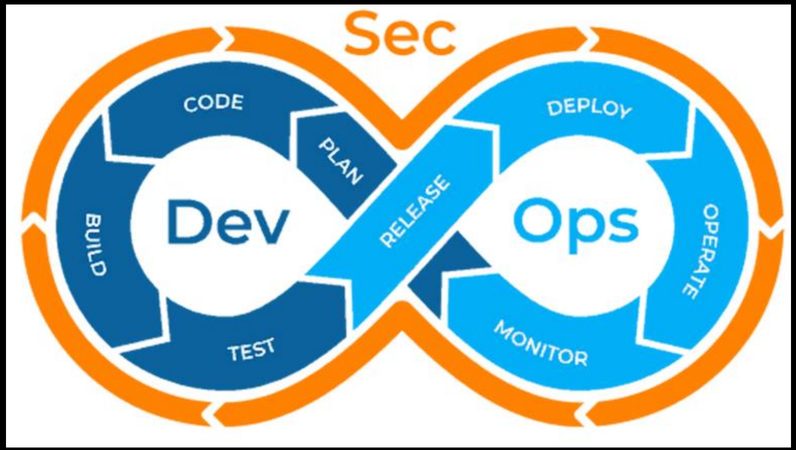
- FFmpeg: CVE-2018-14394, CVE-2020-22016
- Giflib: CVE-2019-15133
- FLAC: CVE-2020-0499, CVE-2021-0561

What other interesting things can we find in binary files...

They said: DevSecOps, CI/CD Pipelines ...sure? ☺

We expect that big vendors use SSDLC and CI/CD Pipeline:

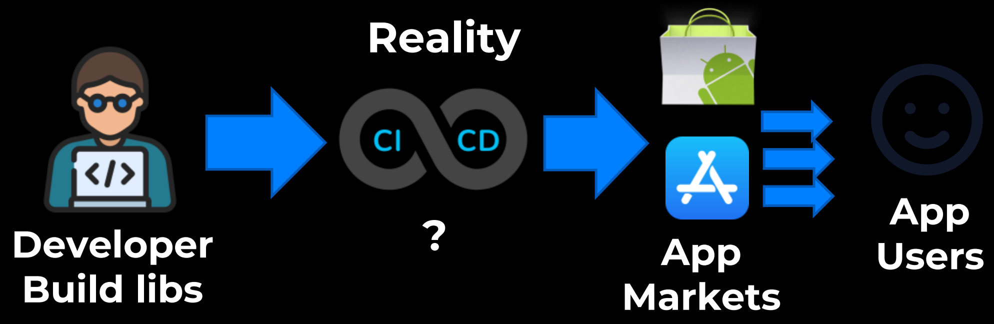
Expectation



We analyze binaries artefacts and see...

They said: DevSecOps, CI/CD Pipelines ...sure? ☺

We analyze binaries artefacts and see:



aivanov – is it sample, in this case it was more seldom last name

/home/**aivanov***/Projects/**company**/android-cloud-sdk/sdk-lib/cloud-sdk/api/src/node_repository/FlatNodeCursor.cpp

--prefix=/Users/ivan*/Development/Projects/Android/comp*/company-android*/media/jni/Tools/ffmpeg/android/arm64-v8a --arch=arm64 --cc=/Users/ivan/Development/SDKs/android-ndk-r15c/toolchains/aarch64-linux-android-4.9/prebuilt/darwin-x86_64/bin/aarch64-linux-android-gcc --pkg-config=/usr/local/bin/pkg-config --enable-gpl --enable-version3 --enable-shared --disable-static --disable-debug --disable-programs --disable-doc --disable-avdevice --disable-swresample --disable-avfilter --disable-everything --enable-libx264 --enable-encoder=libx264 --enable-decoder=mpeg4 --enable-decoder=h264 --enable-demuxer=h264 --enable-demuxer=rtp --enable-muxer=mp4 --enable-parser=h264 --enable-parser=mpeg4video --enable-protocol=file --disable-symver --enable-memalign-hack --enable-asm --cross-prefix=/Users/**ivan***/Development/SDKs/android-ndk-r15c/toolchains/aarch64-linux-android-4.9/prebuilt/darwin-x86_64/bin/aarch64-linux-android- --target-os=linux --enable-cross-compile --sysroot=/Users/**ivan***/Development/SDKs/android-ndk-r15c/p

* - was changed from real for privacy

MacOS?! This is hardly a CI/CD pipeline

It doesn't look like it was built in CI\CD Pipelines ☺ Looks like a build on a developer workstation

Results: How to make your native libs of Android Apps more secure?

OFF ZONE 2024

1. Conduct bug bounties
2. Monitor known vulnerabilities (classical SCA) and bugs/issues, check PoCs for your dependencies. Analyze Silent Vulnerability Fixes
3. Update your vulnerable native libs
4. Fuzzing your libraries and your dependencies
5. Delete service info (build artefacts) from binary files (last name of your developers and build paths)

Next Steps

- Improve commit detection approach (not only using literals), use deeper analysis to detect functions with CVE
- Automation to detect CVE fix in functions
- Publish the implemented BSCA system as a web service
- Continue searching for Silent Vulnerability Fix using LLM with information from Data-Flow Analysis
- Detect versions of packages from APK (Fresco, exoplayer2, ..)
- Try to automate PoC generation using selective function fuzzing

I am open to any help and ideas


OFF ZONE
2024
Q&A
Evgeny Zhukovsky
Founder DAP Solutions
@spb_zhuk