Successfully Fuzzing High Value Targets With Low Tech Strategies

Marc Schönefeld

CanSecWest 2024

Successfully **Fuzzing** High Value Targets With **Low Tech** Strategies

Marc Schönefeld

CanSecWest 2024

Low Tech Fuzzing

Marc Schönefeld

CanSecWest 2024

Agenda

- **Intro**
- Motivation for Low Tech Fuzzing
- Examples
- Lessons Learned
- Closing Thoughts

The Speaker

- Infosec since +20y
- Starting at Blackhat 2002 with "Security Aspects of Java Bytecode Engineering"
- Chromium Hall of Fame, etc.
- Former Red Hat Security Team, found numerous Linux and JDK issues ("B0rken Fonts" at CSW 2011)
- Now working for Oracle Java Team, hunting/handling bugs in JDK and related products
- Hobbyist reversing/hunting bugs, latest: CVE-2024-23300 (GarageBand)

Low Tech Fuzzing

- In this context fuzzing without instrumentation
- Prefer just bit/byte mutation as major fuzzing method   (less splice/trim)
- Typical example is the zzuf fuzzer, or AFL in dumb mode
- Scale with CPU speed and throughput instead of tool complexity
- Have minimal setup time, don’t worry about configurations

Motivation, why Low Tech Fuzzing?

- Not every platform allows sophisticated runtime instrumentation to achieve coverage
- Coverage can be provided ahead-of time due to diversity within a corpus
- PoC testing strategies (with advanced tech level):
  - Enumerating files
    - from a previous fuzzing campaign
    - A downloaded corpus
  - For-loop over seed
    - Mutating each file of a corpus
  - Nested For-loop over seed and density
  - Nested For-loops over seed, density and ranges

Focus

Focus mainly on common crypto formats: X509, PKCS

- Many fuzzing corpora available
  - OpenSSL, BoringSSL, GnuTLS
- More sources to edge case files
  - Frankencerts
    - creating synthetic SSL certificates, by random mutation of parts of real certificates
  - Project Wycheproof (Google)
    - Test collection for many crypto anomalies, artifacts can be reused
- Similar applies to media formats

OpenSSL

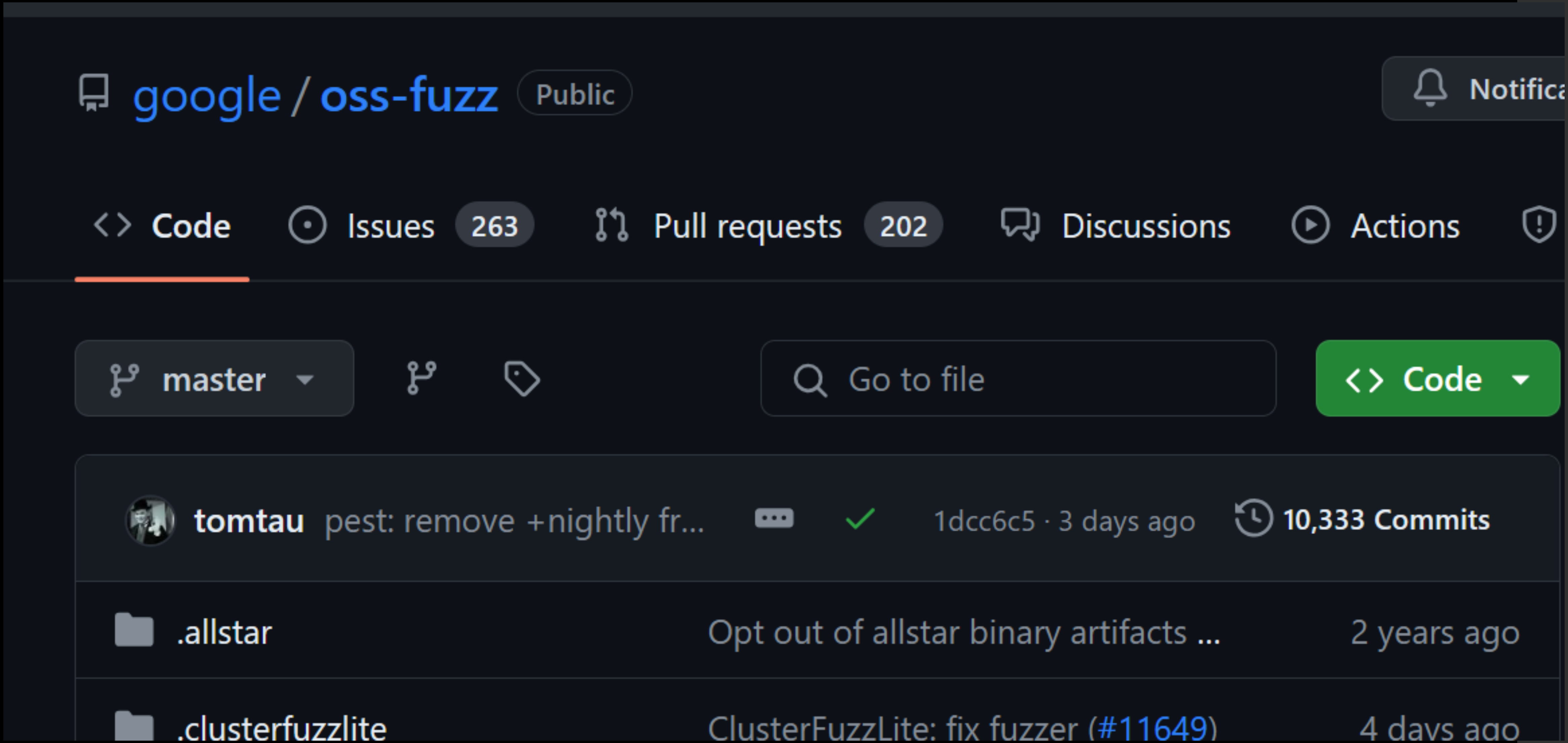
- OpenSSL is a software library that provides secure communications over computer networks. It contains an open-source implementation of the TLS protocols.
- The core library, written in C programming language, implements basic cryptographic functions and provides various utility functions. OpenSSL is widely used by server applications, including the majority of HTTPS websites.
- OpenSSL also includes a rich variety of command-line utilities. The “openssl” tool is a cryptography library that implements the TLS network protocols. It contains different subcommands for any TLS communications needs.
- OpenSSL often embedded in other software products (NodeJS, Android apps,…), problems been discussed in CSW 2018 “Grandma” talk

Fuzzing at scale with OSS-Fuzz

- OSS-Fuzz
  - is a continuous fuzzing service for open-source software, aimed at making common open-source software more secure and stable.
  - uncovers programming errors in software, many of which, like buffer overflow, can have serious security implications.
  - has, since its launch, become a critical service for the open-source community, detecting problems in memory-safe languages such as Go, Rust, and Python

Fuzzing OpenSSL in OSS-Fuzz

- OpenSSL repo contains various harnesses and corpora for fuzzing various functionality:
  - CMS (message signing)
  - X509 (certificates)
  - ASN1 (DER/PEM)
  - …
- OSSFuzz uses these corpora when fuzzing OpenSSL directly
- Unfortunately, OSSFuzz does not fuzz embedded OpenSSL use
  - as in Node.js

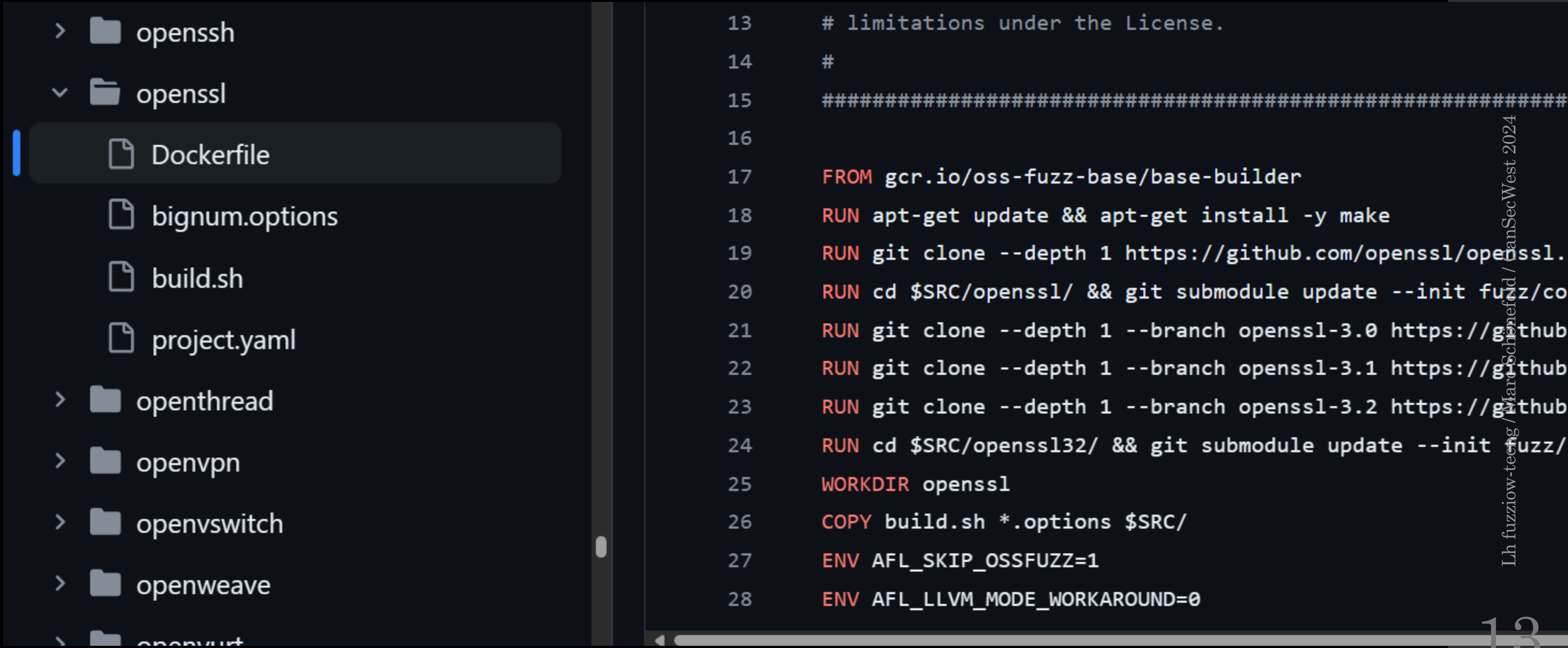


google / oss-fuzz Public

tomtau pest: remove +nightly fr... 1dcc6c5 · 3 days ago 10,333 Commits

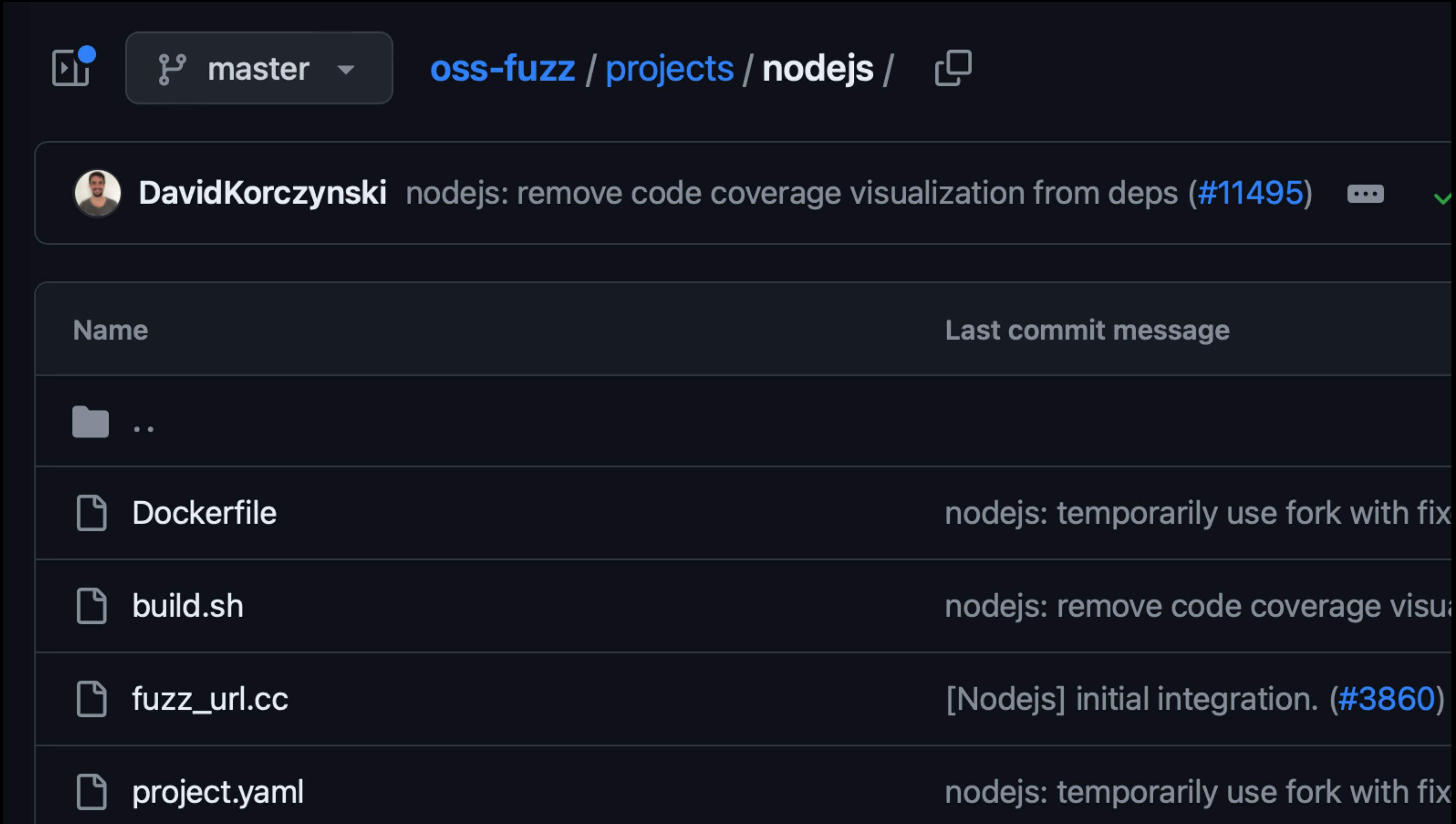
.allstar	Opt out of allstar binary artifacts ...	2 years ago
.clusterfuzzlite	ClusterFuzzLite: fix fuzzer (#11649)	4 days ago

Fuzzing OpenSSL with OSSFuzz



```
# limitations under the License.
#
##########################################################
 
FROM gcr.io/oss-fuzz-base/base-builder
RUN apt-get update && apt-get install -y make
RUN git clone --depth 1 https://github.com/openssl/openssl.
RUN cd $SRC/openssl/ && git submodule update --init fuzz/co
RUN git clone --depth 1 --branch openssl-3.0 https://github
RUN git clone --depth 1 --branch openssl-3.1 https://github
RUN git clone --depth 1 --branch openssl-3.2 https://github
RUN cd $SRC/openssl32/ && git submodule update --init fuzz/
WORKDIR openssl
COPY build.sh *.options $SRC/
ENV AFL_SKIP_OSSFUZZ=1
ENV AFL_LLVM_MODE_WORKAROUND=0
```

Fuzzing Node.js with OSSFuzz



master

oss-fuzz / projects / nodejs /

DavidKorczynski nodejs: remove code coverage visualization from deps (#11495)

Name	Last commit message
..	
Dockerfile	nodejs: temporarily use fork with fix
build.sh	nodejs: remove code coverage visu
fuzz_url.cc	[Nodejs] initial integration. (#3860)
project.yaml	nodejs: temporarily use fork with fix

Example 1: CVE-2022-4450

CVE-2022-4450 What was the bug?

- The function PEM_read_bio_ex() reads a PEM file from a BIO and parses and decodes the "name" (e.g. "CERTIFICATE").
- In the event of a failure in PEM_read_bio_ex() OpenSSL frees, but not clears the pointers stored in *header and *data.
- Since, on success, the caller is responsible for freeing these ptrs this can potentially lead to a double free if the caller frees them even on failure.
- This could be exploited by an attacker who can supply malicious PEM files for parsing.
- The OpenSSL asn1parse command line application is also impacted by this issue.
- OpenSSL was affected since 3.0.0, and fixed in OpenSSL 3.0.8

CVE-2022-4450
What is PEM?

- PEM (Privacy Enhanced Mail) is a widely used container file format for storing and sending cryptographic keys, certificates, and other data (RFC 7468).
- PEM files containing one or more crypto items in Base64 ASCII encoding, each with plain-text headers and footers (e.g

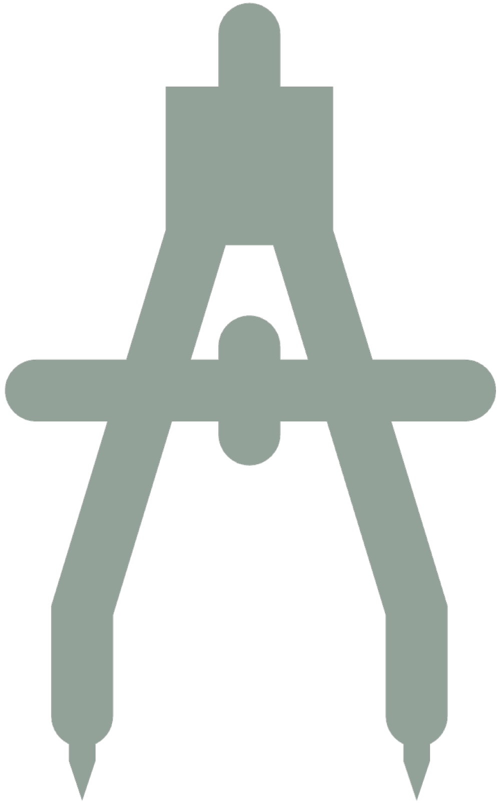
  `-----BEGIN CERTIFICATE-----`
  and
  `-----END CERTIFICATE-----`
).
- A single PEM file can contain an end-entity certificate, a private key, or multiple certificates forming a complete chain of trust (as with PKCS7).

CVE-2022-4450: What is PEM_read_bio_ex good for?

- The PEM_read_bio_ex() function is used to read PEM formatted data from an input BIO (Basic Input Output).

- The function takes (among others) the following parameters:
  - BIO *in: A pointer to the input BIO.
  - char **name: A pointer to a string where the name of the type of contained data will be stored.
  - char **header: A pointer to a string where the header information will be stored.

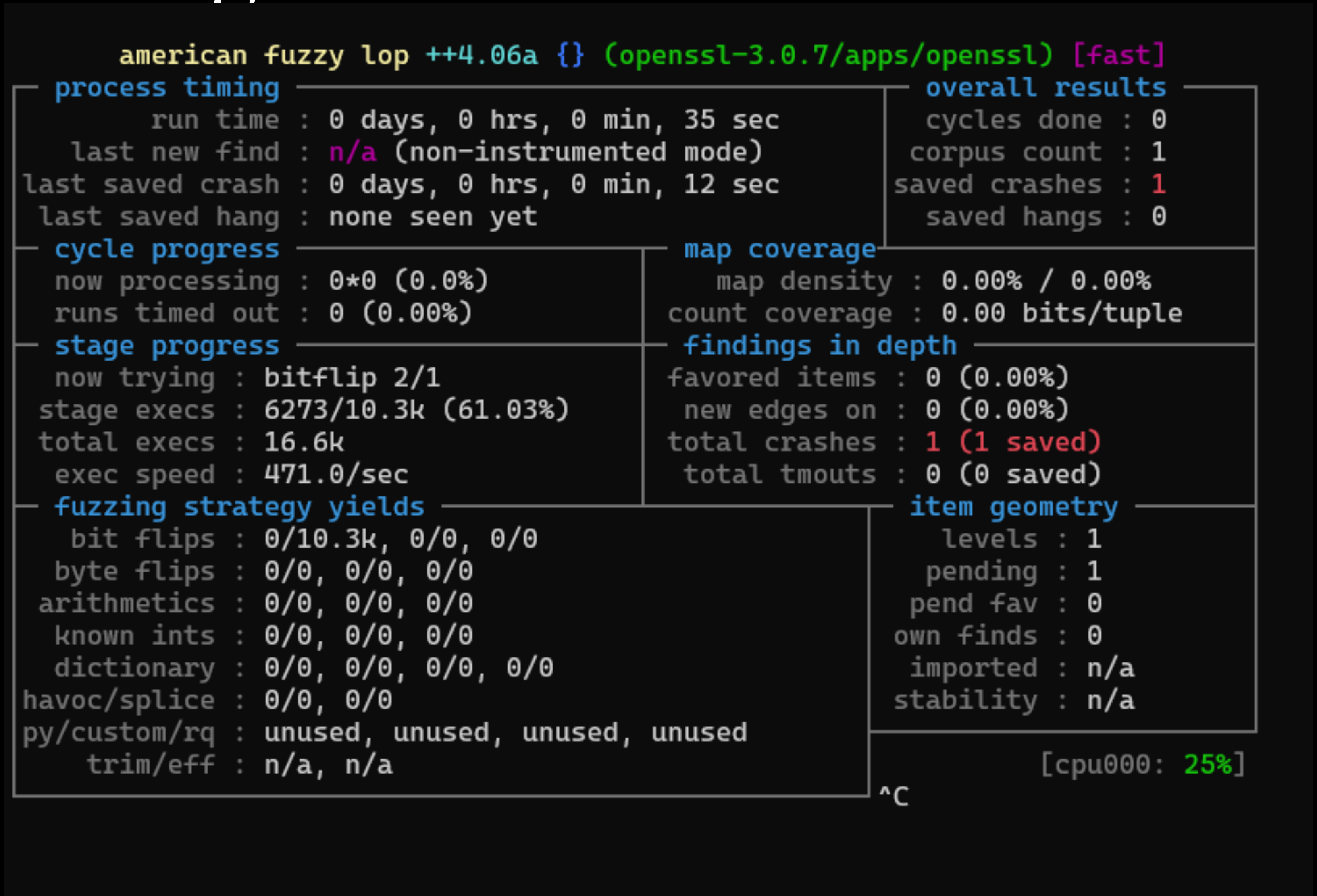
- This function is typically used when reading PEM structures from files or network connections.



CVE-2022-4450
Our low tech fuzzing setup

- Traverse the X509 artifacts in the corresponding OpenSSL corpus
- This corpus comes with DER and PEM artifacts, so convert entries to both formats
- Bit-Mutate each file without instrumentation using afl-fuzz (-n –D), could also use zzuf
- Feed the result to the verify command of the ‘openssl’ tool (optionally use an ASAN build)
- Run in an endless loop and wait for crashes

CVE-2022-4450
Running dumb with AFL



```
       american fuzzy lop ++4.06a {} (openssl-3.0.7/apps/openssl) [fast]
┌─ process timing ───────────────────────────────────┬─ overall results ────┐
│        run time : 0 days, 0 hrs, 0 min, 35 sec     │  cycles done : 0     │
│   last new find : n/a (non-instrumented mode)      │ corpus count : 1     │
│last saved crash : 0 days, 0 hrs, 0 min, 12 sec     │saved crashes : 1     │
│ last saved hang : none seen yet                    │  saved hangs : 0     │
├─ cycle progress ────────────────────┬─ map coverage┴──────────────────────┤
│  now processing : 0*0 (0.0%)        │    map density : 0.00% / 0.00%      │
│  runs timed out : 0 (0.00%)         │ count coverage : 0.00 bits/tuple    │
├─ stage progress ────────────────────┼─ findings in depth ─────────────────┤
│  now trying : bitflip 2/1           │ favored items : 0 (0.00%)           │
│ stage execs : 6273/10.3k (61.03%)   │  new edges on : 0 (0.00%)           │
│ total execs : 16.6k                 │ total crashes : 1 (1 saved)         │
│  exec speed : 471.0/sec             │  total tmouts : 0 (0 saved)         │
├─ fuzzing strategy yields ───────────┴─────────────┬─ item geometry ───────┤
│   bit flips : 0/10.3k, 0/0, 0/0                   │    levels : 1         │
│  byte flips : 0/0, 0/0, 0/0                       │   pending : 1         │
│ arithmetics : 0/0, 0/0, 0/0                       │  pend fav : 0         │
│  known ints : 0/0, 0/0, 0/0                       │ own finds : 0         │
│  dictionary : 0/0, 0/0, 0/0, 0/0                  │  imported : n/a       │
│havoc/splice : 0/0, 0/0                            │ stability : n/a       │
│py/custom/rq : unused, unused, unused, unused      ├───────────────────────┘
│    trim/eff : n/a, n/a                            │          [cpu000: 25%]
└───────────────────────────────────────────────────┘^C
```

CVE-2022-4450
In the debugger

-
  ```
  Starting program:
  openssl verify  -CAfile Starting program:
  /home/user/cert_verify/ec/openssl-3.0.7/apps/openssl verify -CAfile
  crashes/id:000000,sig:06,src:000000,time:23165,execs:10514,op:flip2,p
  os:28
  free(): double free detected in tcache 2
  ```

- `Program received signal SIGABRT, Aborted.`

- `0x000000000079daac in pthread_kill ()`

- `(gdb) bt`

-
  ```
  #0  0x000000000079daac in pthread_kill ()
  #1  0x00000000007837d6 in raise ()
  #2  0x00000000004022ab in abort ()
  #3  0x00000000007977e6 in __libc_message ()
  #4  0x000000000079e39c in malloc_printerr ()
  #5  0x000000000079fb48 in _int_free ()
  #6  0x00000000007a2b11 in free ()
  #7  0x00000000005cae1a in PEM_X509_INFO_read_bio_ex ()
  #8  0x000000000060f268 in X509_load_cert_crl_file_ex.part.0 ()
  #9  0x000000000060f695 in by_file_ctrl_ex ()
  #10 0x000000000045ef31 in setup_verify ()
  #11 0x0000000000457057 in verify_main ()
  #12 0x0000000000427b12 in do_cmd ()
  #13 0x000000000040303f in main ()
  ```

CVE-2022-4450
The cause

- A minimal bitflip causes the crash
- xxd orig_ca

-
  ```
  2d2d 2d2d 2d42 4547 494e 2043 4552 5449  -----BEGIN CERTI
  4649 4341 5445 2d2d 2d2d 2d0a 4d49 4944  FICATE-----.MIID
  6954 4343 416e 4767 4177 4942 4167 4955  iTCCAnGgAwIBAgIU
  ```
- xxd id\:000000\, ..
  \:23165\,execs\:10514\,op\:flip2\,pos\:28

-
  ```
  2d2d 2d2d 2d42 4547 494e 2043 4552 5449  -----BEGIN CERTI
  4649 4341 5445 2d2d 2d2d 2d0a 2d49 4944  FICATE-----.-IID
  6954 4343 416e 4767 4177 4942 4167 4955  iTCCAnGgAwIBAgIU
  ```

CVE-2022-4450 A minimal bitflip causes a null ASN.1 sequence

- This causes the ASN.1 representation to differ in the first sequence set to zero:

- Original:

```
<     0:d=0  hl=4 l= 905 cons: SEQUENCE
<     4:d=1  hl=4 l= 625 cons: SEQUENCE
<     8:d=2  hl=2 l=   3 cons: cont [ 0 ]
<    10:d=3  hl=2 l=   1 prim: INTEGER           :02
<    13:d=2  hl=2 l=  20 prim: INTEGER
:6FEB65DFDC5A63FAB80BFC4501ABCAD53C91ABE0
```

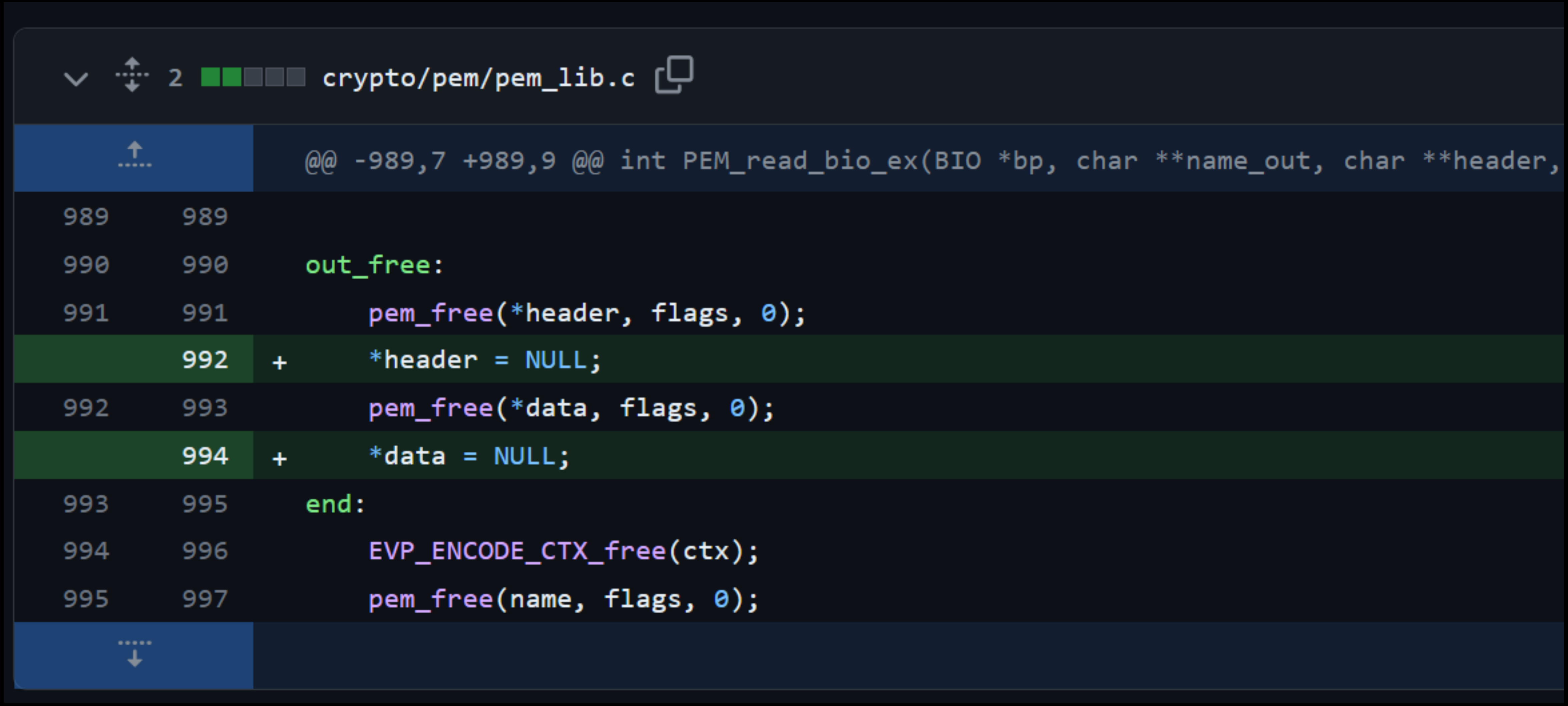
- Fuzzed:

```
>     0:d=0  hl=2 l=   0 prim: NULL
>     2:d=0  hl=2 l=  84 cons: SEQUENCE
>     4:d=1  hl=2 l=  11 cons: SET
>     6:d=2  hl=2 l=   9 cons: SEQUENCE
```

CVE-2022-4450 What was the fix strategy?

- The pointers to store header and data information were not reset to null when the buffer they point to was freed.
- This occurred in several places.
- Fix idea: prior to releasing the buffer, also clear the internal pointer to the buffer, which prevents the double-free.

CVE-2022-4450
What was the fix?

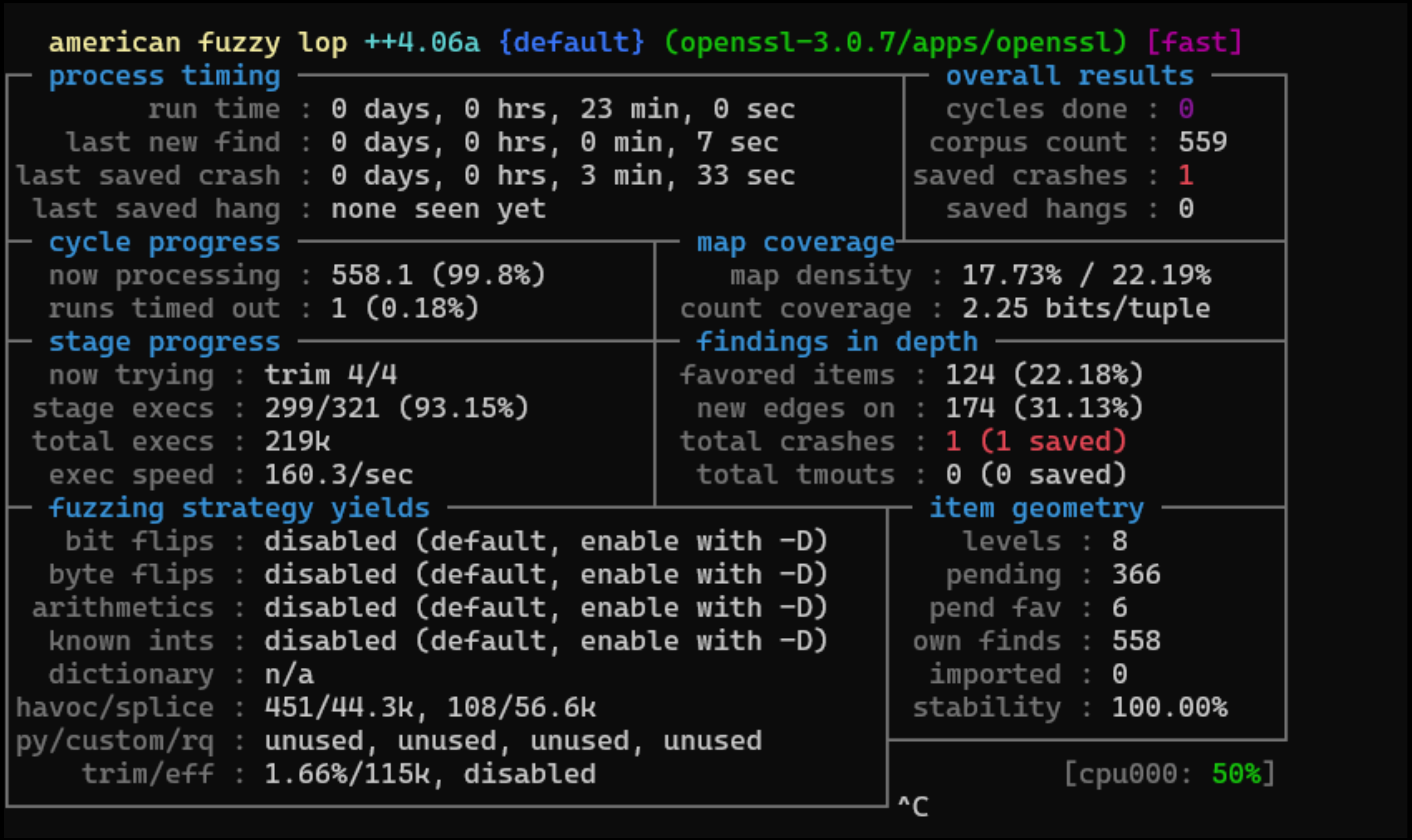


2 🟩🟩⬛⬛⬛ crypto/pem/pem_lib.c

```diff
@@ -989,7 +989,9 @@ int PEM_read_bio_ex(BIO *bp, char **name_out, char **header,

 out_free:
     pem_free(*header, flags, 0);
+    *header = NULL;
     pem_free(*data, flags, 0);
+    *data = NULL;
 end:
     EVP_ENCODE_CTX_free(ctx);
     pem_free(name, flags, 0);
```

CVE-2022-4450
Running instrumented with AFL



```
  american fuzzy lop ++4.06a {default} (openssl-3.0.7/apps/openssl) [fast]
┌─ process timing ────────────────────────────────────┬─ overall results ────┐
│        run time : 0 days, 0 hrs, 23 min, 0 sec      │  cycles done : 0     │
│   last new find : 0 days, 0 hrs, 0 min, 7 sec       │ corpus count : 559   │
│last saved crash : 0 days, 0 hrs, 3 min, 33 sec      │saved crashes : 1     │
│ last saved hang : none seen yet                     │  saved hangs : 0     │
├─ cycle progress ─────────────────────┬─ map coverage┴──────────────────────┤
│  now processing : 558.1 (99.8%)      │    map density : 17.73% / 22.19%   │
│  runs timed out : 1 (0.18%)          │ count coverage : 2.25 bits/tuple   │
├─ stage progress ─────────────────────┼─ findings in depth ─────────────────┤
│  now trying : trim 4/4               │ favored items : 124 (22.18%)        │
│ stage execs : 299/321 (93.15%)       │  new edges on : 174 (31.13%)        │
│ total execs : 219k                   │ total crashes : 1 (1 saved)         │
│  exec speed : 160.3/sec              │  total tmouts : 0 (0 saved)         │
├─ fuzzing strategy yields ────────────┴─────────────┬─ item geometry ───────┤
│   bit flips : disabled (default, enable with -D)   │    levels : 8         │
│  byte flips : disabled (default, enable with -D)   │   pending : 366       │
│ arithmetics : disabled (default, enable with -D)   │  pend fav : 6         │
│  known ints : disabled (default, enable with -D)   │ own finds : 558       │
│  dictionary : n/a                                  │  imported : 0         │
│havoc/splice : 451/44.3k, 108/56.6k                 │ stability : 100.00%   │
│py/custom/rq : unused, unused, unused, unused       ├───────────────────────┘
│    trim/eff : 1.66%/115k, disabled                 │          [cpu000: 50%]
└────────────────────────────────────────────────────┘^C
```

CVE-2022-4450 Timeline

- Reported Dec 27,2022
- No confirmation mail
- Unknown when patch was ready
- Fixed in OpenSSL 3.0.8, Feb 7, 2023
- Got added to advisory as of Feb 21, 2023

Example 2: CVE-2023-0216

CVE-2023-0216 What was the bug?

- An invalid pointer dereference on read can be triggered when an application tries to load malformed PKCS7 data with the d2i_PKCS7(), d2i_PKCS7_bio() or d2i_PKCS7_fp() functions.
- The result of the dereference is an application crash which could lead to a denial-of-service attack.
- The TLS implementation in OpenSSL does not call this function however third-party applications might call these functions on untrusted data.
- OpenSSL was affected since 3.0.0, and fixed in OpenSSL 3.0.8

CVE-2023-0216
What is PKCS7?

- PKCS #7, also known as Cryptographic Message Syntax (CMS), is a standard syntax for storing signed and/or encrypted data.
- It is part of the family of standards called Public-Key Cryptography Standards (PKCS), created by RSA Laboratories.
- A typical use of a PKCS #7 file would be to store certificates and/or certificate revocation lists (CRL).

CVE-2023-0216:
What is d2i_PKCS7 good for?

-
  ```
  PKCS7 *d2i_PKCS7(
     PKCS7 **val_out,
     const unsigned char **der_in,
     long length).
  ```

- The function creates a PKCS#7 structure from DER formatted data, takes a pointer to a buffer containing the DER encoded PKCS#7 structure, the length of this buffer, and a pointer to a PKCS7 structure.

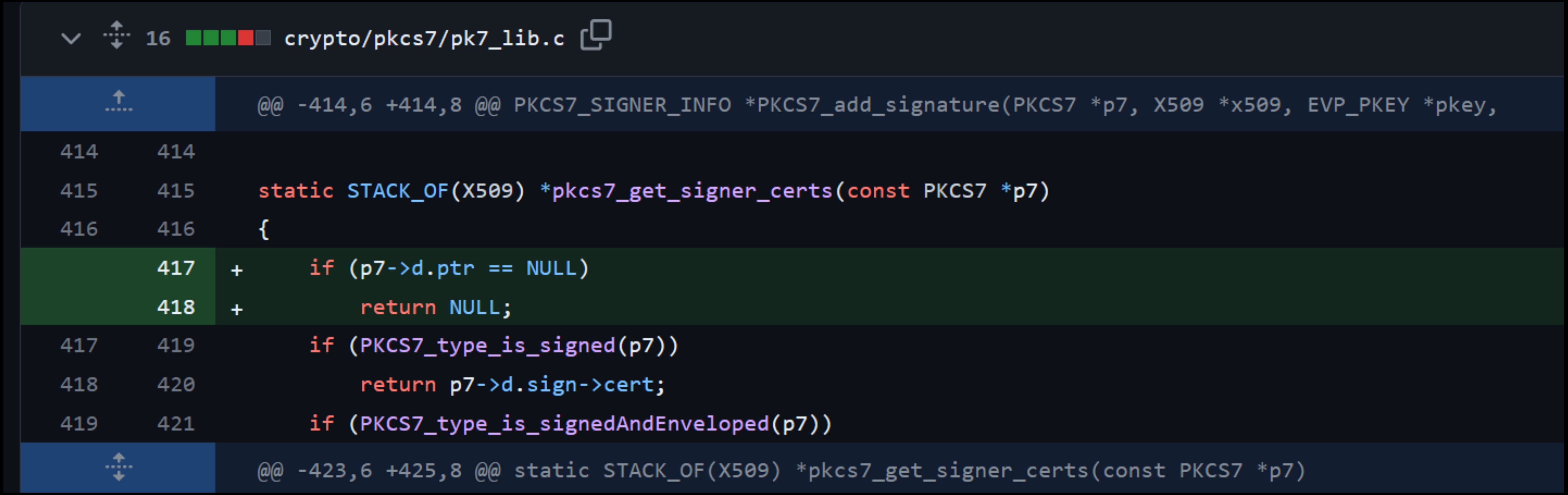
- If the val_out argument is not a NULL pointer, the PKCS7 structure is written to *val_out. If *val_out is NULL, a new PKCS7 structure is created and *val_out is updated to point to it.

- Returns a pointer to the PKCS7 structure on success, or NULL if an error occurred.

CVE-2023-0216 What was the fix strategy?

- The PKCS7 data element to store the binary raw data (d.ptr) was not checked for sanity.
- This occurred in several places.
- Fix idea: prior to further processing the PKCS7 structure, the value of d.ptr is validated

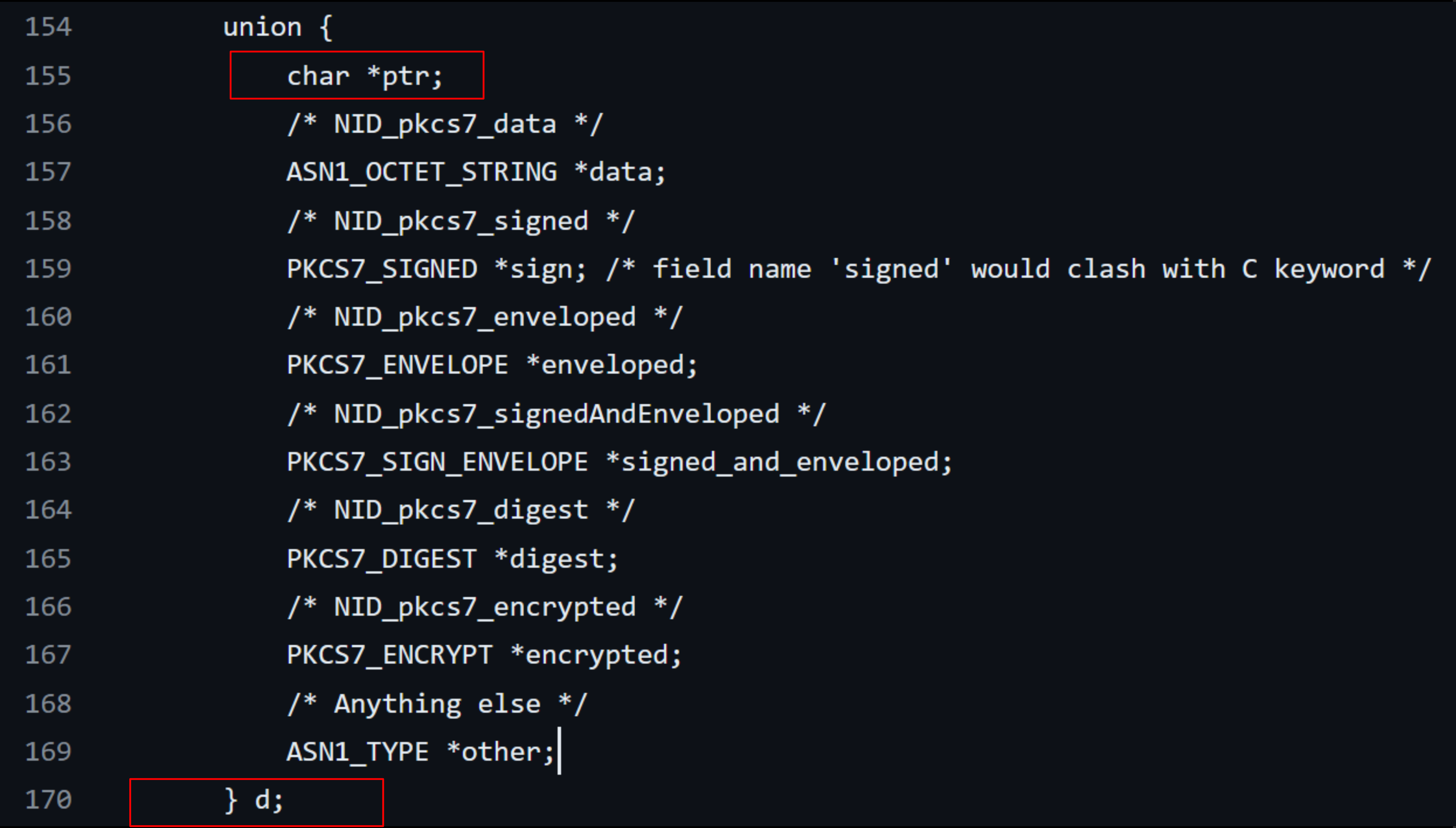
CVE-2023-0216
What was the fix?



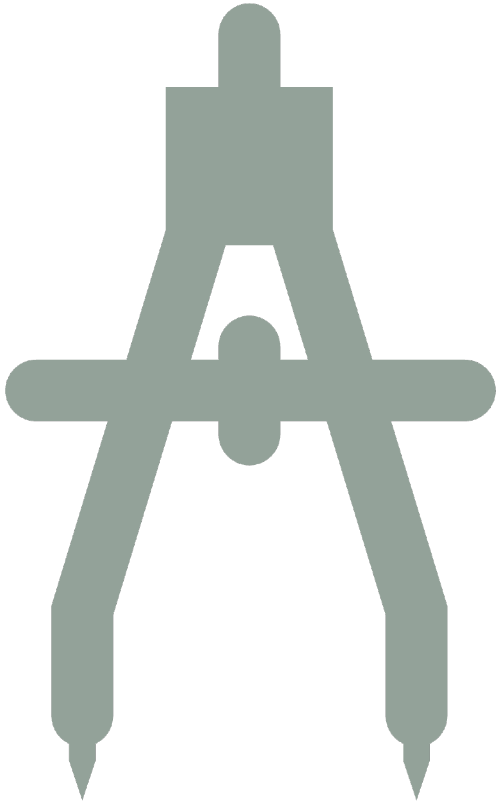
```
16 🟩🟩🟩🟥⬛ crypto/pkcs7/pk7_lib.c

@@ -414,6 +414,8 @@ PKCS7_SIGNER_INFO *PKCS7_add_signature(PKCS7 *p7, X509 *x509, EVP_PKEY *pkey,

 static STACK_OF(X509) *pkcs7_get_signer_certs(const PKCS7 *p7)
 {
+    if (p7->d.ptr == NULL)
+        return NULL;
     if (PKCS7_type_is_signed(p7))
         return p7->d.sign->cert;
     if (PKCS7_type_is_signedAndEnveloped(p7))
@@ -423,6 +425,8 @@ static STACK_OF(X509) *pkcs7_get_signer_certs(const PKCS7 *p7)
```



```
    union {
        char *ptr;
        /* NID_pkcs7_data */
        ASN1_OCTET_STRING *data;
        /* NID_pkcs7_signed */
        PKCS7_SIGNED *sign; /* field name 'signed' would clash with C keyword */
        /* NID_pkcs7_enveloped */
        PKCS7_ENVELOPE *enveloped;
        /* NID_pkcs7_signedAndEnveloped */
        PKCS7_SIGN_ENVELOPE *signed_and_enveloped;
        /* NID_pkcs7_digest */
        PKCS7_DIGEST *digest;
        /* NID_pkcs7_encrypted */
        PKCS7_ENCRYPT *encrypted;
        /* Anything else */
        ASN1_TYPE *other;
    } d;
```



CVE-2023-0216
Our low tech fuzzing setup

- Traverse artifacts in the OpenSSL subcorpora
- Feed each to the pkcs7 command of the ‘openssl’ tool (optionally use an ASAN build)
- Run in an endless loop and wait for crashes

CVE-2023-0216
Our low tech fuzzing setup

-
  ```
  >find fuzz/corpora/cms/ -type f  | xargs -t -n1 apps/openssl pkcs7
  -inform der -noout  -in

  apps/openssl pkcs7 -inform der -noout -in
  fuzz/corpora/cms/c1682be3e45f36fc45625d10e9bd21df126a4b1a
  ```

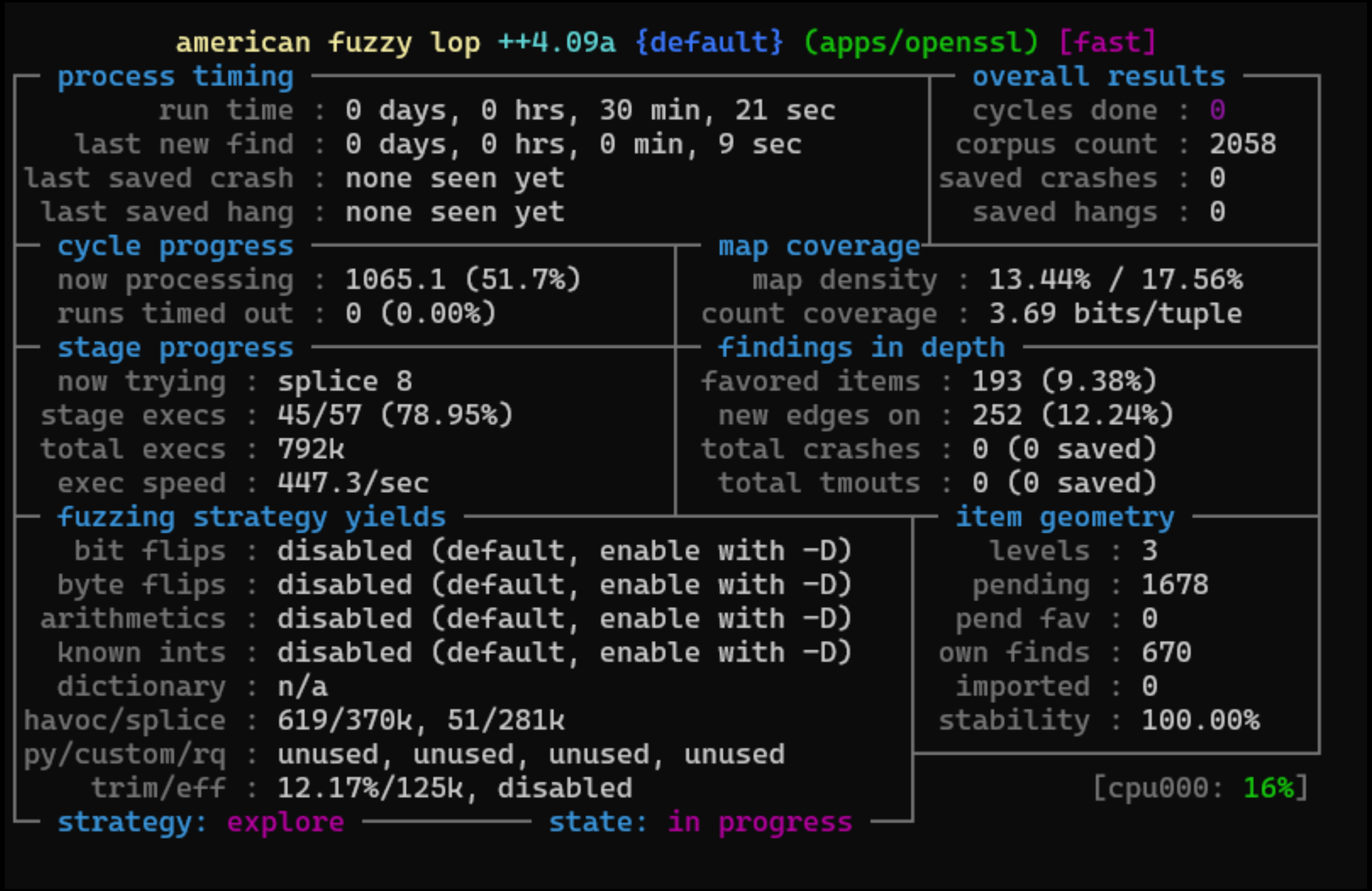
-
  ```
  unable to load PKCS7 object
  00000000:error:0680007B:asn1 encoding
  routines:ASN1_get_object:header too long:crypto/asn1/asn1_lib.c:105
  ```

- [..after a few files..]

-
  ```
  apps/openssl pkcs7 -inform der -noout -in
  fuzz/corpora/cms/2efd07909f95d84de40ebb8b2bc8f3d734939f2d
  ```

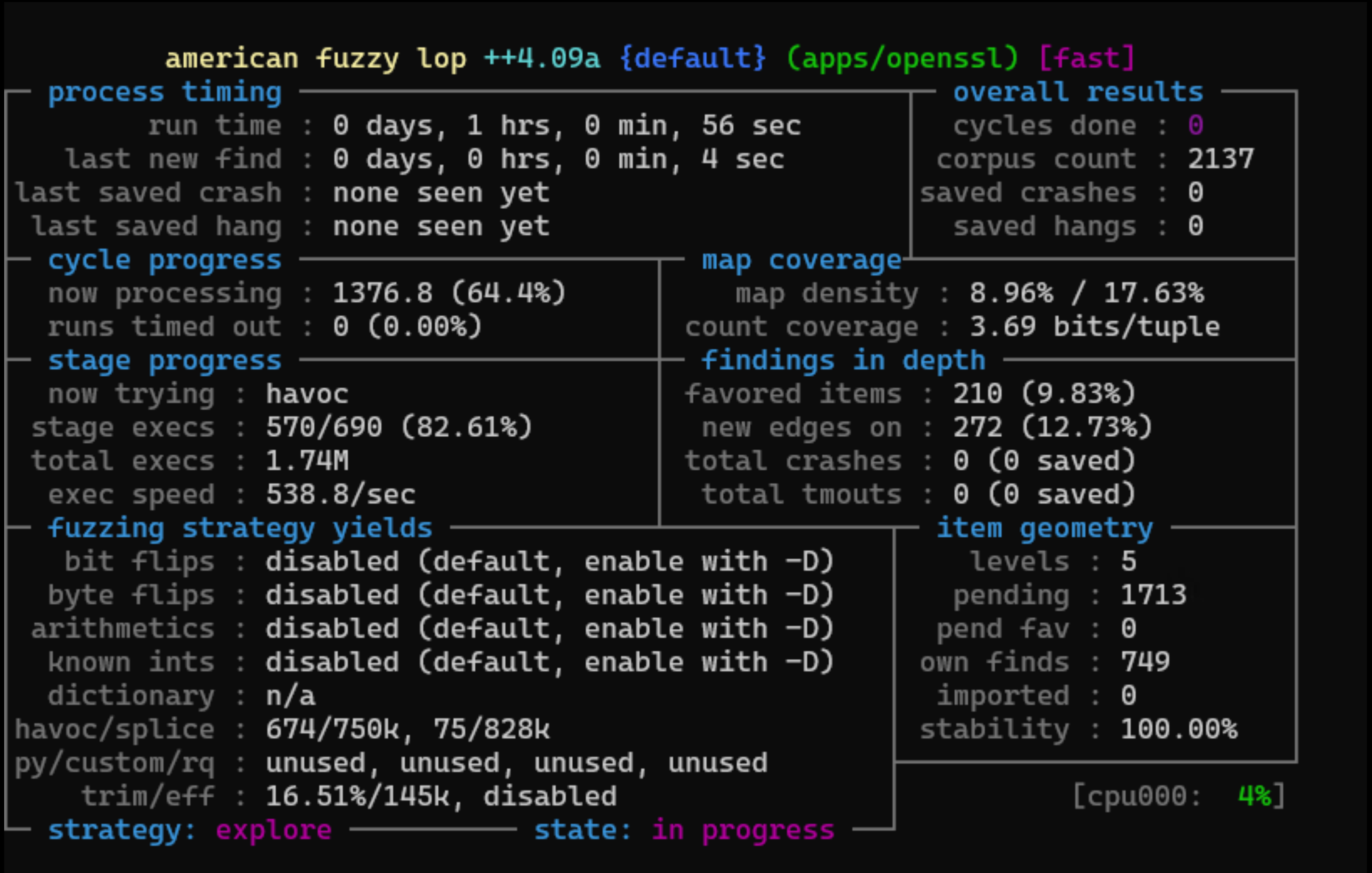
-
  ```
  xargs: apps/openssl: terminated by signal 11
  ```

AFL (in Qemu mode)
no crash, after 30 minutes



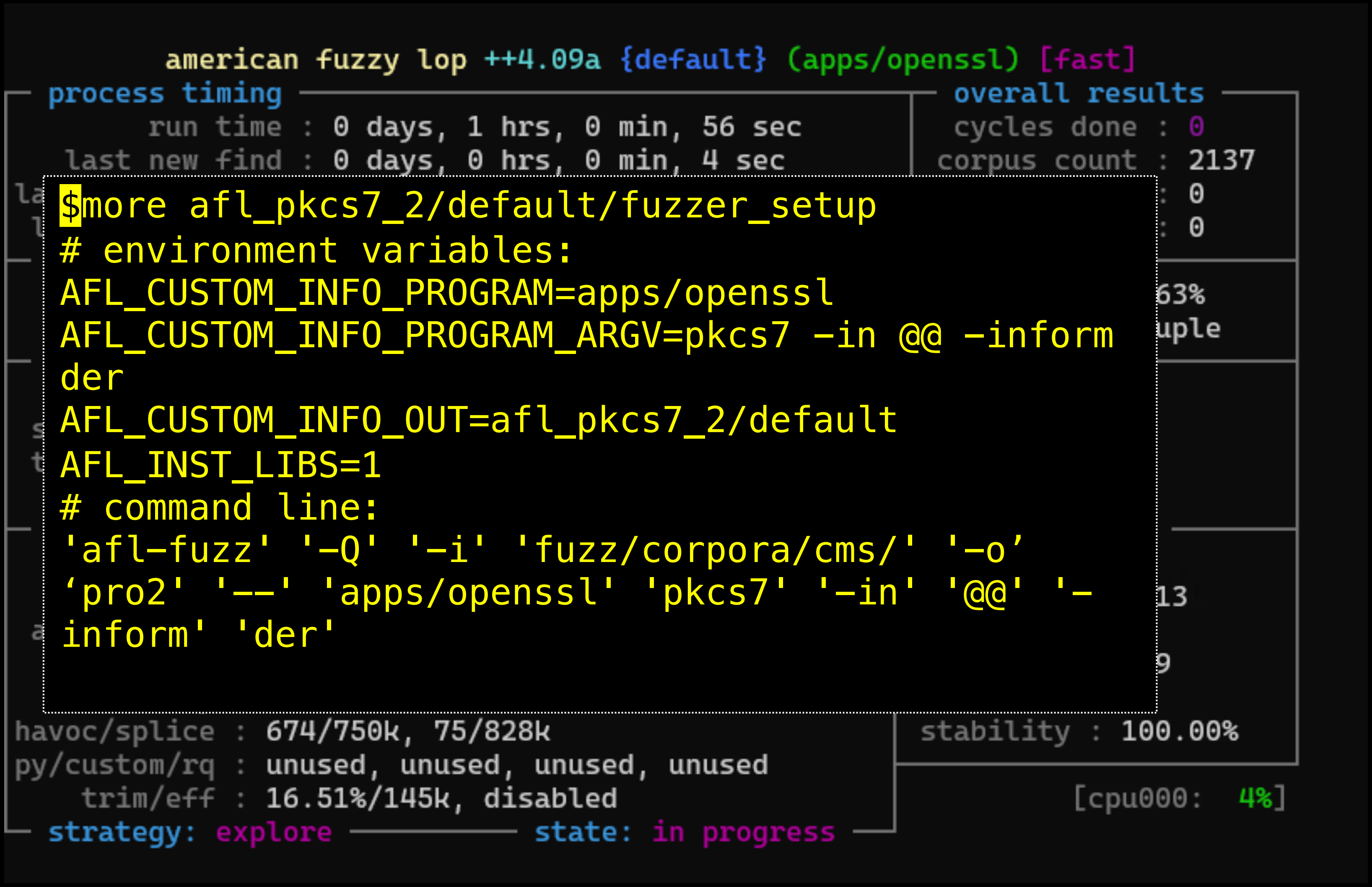
```
          american fuzzy lop ++4.09a {default} (apps/openssl) [fast]
┌─ process timing ─────────────────────────────────────┬─ overall results ──┐
│        run time : 0 days, 0 hrs, 30 min, 21 sec      │  cycles done : 0    │
│   last new find : 0 days, 0 hrs, 0 min, 9 sec        │ corpus count : 2058 │
│last saved crash : none seen yet                      │saved crashes : 0    │
│ last saved hang : none seen yet                      │  saved hangs : 0    │
├─ cycle progress ──────────────────────┬─ map coverage┴─────────────────────┤
│  now processing : 1065.1 (51.7%)      │    map density : 13.44% / 17.56%   │
│  runs timed out : 0 (0.00%)           │ count coverage : 3.69 bits/tuple   │
├─ stage progress ──────────────────────┼─ findings in depth ────────────────┤
│  now trying : splice 8                │ favored items : 193 (9.38%)        │
│ stage execs : 45/57 (78.95%)          │  new edges on : 252 (12.24%)       │
│ total execs : 792k                    │ total crashes : 0 (0 saved)        │
│  exec speed : 447.3/sec               │  total tmouts : 0 (0 saved)        │
├─ fuzzing strategy yields ─────────────┴─────────────┬─ item geometry ──────┤
│   bit flips : disabled (default, enable with -D)    │    levels : 3        │
│  byte flips : disabled (default, enable with -D)    │   pending : 1678     │
│ arithmetics : disabled (default, enable with -D)    │  pend fav : 0        │
│  known ints : disabled (default, enable with -D)    │ own finds : 670      │
│  dictionary : n/a                                   │  imported : 0        │
│havoc/splice : 619/370k, 51/281k                     │ stability : 100.00%  │
│py/custom/rq : unused, unused, unused, unused        ├──────────────────────┘
│    trim/eff : 12.17%/125k, disabled                 │          [cpu000: 16%]
└─ strategy: explore ────────── state: in progress ───┘
```

AFL (in Qemu mode)
no crash, even after 60 minutes



```
        american fuzzy lop ++4.09a {default} (apps/openssl) [fast]
─ process timing ─────────────────────────────────────┬─ overall results ──┐
        run time : 0 days, 1 hrs, 0 min, 56 sec       │  cycles done : 0   │
   last new find : 0 days, 0 hrs, 0 min, 4 sec        │ corpus count : 2137│
last saved crash : none seen yet                      │saved crashes : 0   │
 last saved hang : none seen yet                      │  saved hangs : 0   │
─ cycle progress ──────────────────────┬─ map coverage┴────────────────────┤
  now processing : 1376.8 (64.4%)      │    map density : 8.96% / 17.63%  │
  runs timed out : 0 (0.00%)           │ count coverage : 3.69 bits/tuple │
─ stage progress ──────────────────────┼─ findings in depth ───────────────┤
  now trying : havoc                   │ favored items : 210 (9.83%)      │
 stage execs : 570/690 (82.61%)        │  new edges on : 272 (12.73%)     │
 total execs : 1.74M                   │ total crashes : 0 (0 saved)      │
  exec speed : 538.8/sec               │  total tmouts : 0 (0 saved)      │
─ fuzzing strategy yields ─────────────┴───────────┬─ item geometry ───────┤
   bit flips : disabled (default, enable with -D) │    levels : 5         │
  byte flips : disabled (default, enable with -D) │   pending : 1713      │
 arithmetics : disabled (default, enable with -D) │  pend fav : 0         │
  known ints : disabled (default, enable with -D) │ own finds : 749       │
  dictionary : n/a                                │  imported : 0         │
havoc/splice : 674/750k, 75/828k                  │ stability : 100.00%   │
py/custom/rq : unused, unused, unused, unused     ├───────────────────────┘
    trim/eff : 16.51%/145k, disabled              │          [cpu000:  4%]
─ strategy: explore ────────── state: in progress ┘
```

AFL (in Qemu mode)
no crash, even after 60 minutes



```
            american fuzzy lop ++4.09a {default} (apps/openssl) [fast]
┌─ process timing ────────────────────────────────────┬─ overall results ──┐
│        run time : 0 days, 1 hrs, 0 min, 56 sec      │  cycles done : 0   │
│   last new find : 0 days, 0 hrs, 0 min, 4 sec       │ corpus count : 2137│
│la                                                   │            : 0     │
│ l                                                   │            : 0     │
├─                                                    ├────────────────────┤
│                                                      63%                 │
│                                                      uple                │
├─                                                    ├────────────────────┤
│ s                                                                        │
│ t                                                                        │
├─                                                    ├────────────────────┤
│                                                      13                  │
│ a                                                                        │
│                                                      9                   │
│havoc/splice : 674/750k, 75/828k                     │ stability : 100.00%│
│py/custom/rq : unused, unused, unused, unused        ├────────────────────┘
│    trim/eff : 16.51%/145k, disabled                 │        [cpu000:  4%]
└─ strategy: explore ────────── state: in progress ───┘
```

```
$more afl_pkcs7_2/default/fuzzer_setup
# environment variables:
AFL_CUSTOM_INFO_PROGRAM=apps/openssl
AFL_CUSTOM_INFO_PROGRAM_ARGV=pkcs7 -in @@ -inform
der
AFL_CUSTOM_INFO_OUT=afl_pkcs7_2/default
AFL_INST_LIBS=1
# command line:
'afl-fuzz' '-Q' '-i' 'fuzz/corpora/cms/' '-o’
‘pro2' '--' 'apps/openssl' 'pkcs7' '-in' '@@' '-
inform' 'der'
```

CVE-2023-0216
In the debugger

```
>gdb --args apps/openssl pkcs7 -in
fuzz/corpora/cms/2efd07909f95d84de40ebb8b2bc8f3d734939f2d -
inform der

--

Program received signal SIGSEGV, Segmentation fault.

0x00000000005de43d in ossl_pkcs7_resolve_libctx ()

(gdb) bt

#0  0x00000000005de43d in ossl_pkcs7_resolve_libctx ()
#1  0x0000000000638ee5 in d2i_PKCS7_bio ()
#2  0x000000000042cebe in pkcs7_main ()
#3  0x0000000000427b12 in do_cmd ()
#4  0x000000000040303f in main ()
```

CVE-2023-0216
In the debugger

```
(gdb)  disass $pc-10,$pc+10

Dump of assembler code from 0x5de433 to 0x5de447:

0x..05de433 <ossl_.._libctx+115>:  and    BYTE PTR [rbp+0x31],al
0x..05de436 <ossl_.._libctx+118>:  in     eax,dx
0x..05de437 <ossl_.._libctx+119>:  cmp    QWORD PTR [rsp+0x8],0x0
0x..05de43d <ossl_.._libctx+125>:  mov    rbp,QWORD PTR [rax+0x10]
0x..05de441 <ossl_.._libctx+129>:  jne    0x5de4dd <ossl.._libctx+285>

(gdb) info register rax
rax            0x0                 0
```

CVE-2023-0216
Where did the PoC come from?

OpenSSL‘s own CMS corpus included the PoC since 2018, however likely not tested with PKCS7 functions, despite cms format has PKCS7 under the hood:

```
$ git log fuzz/corpora/cms/2efd07909f95d84de40ebb8b2bc8f3d734939f2d
commit 0f735011962830ceaa9a7ab0b9d91129d9ba011d
Date:   Tue Apr 4 16:15:37 2023 +0200
   Remove fuzz corpora data from the repository
   ..
commit 0b89db6b2acb6cca36f812ba51119927563b3cac
Date:   Wed Aug 22 23:31:01 2018 +0200
    Update fuzz corpora
    ..

$ openssl asn1parse -inform der  -in
fuzz/corpora/cms/2efd07909f95d84de40ebb8b2bc8f3d734939f2d
    0:d=0  hl=2 l=  11 cons: SEQUENCE
    2:d=1  hl=2 l=   9 prim: OBJECT            :pkcs7-signedData
```

CVE-2023-0216 Timeline

- Reported Dec 23,2022
- Confirmation Dec 24,2022
- Patch ready Jan 10, 2023
- Fixed in OpenSSL 3.0.8, Feb 7, 2023

Example 3: CVE-2023-30588

CVE-2023-30588 What was the bug?

- When an invalid public key is used to create an x509 certificate using the crypto.X509Certificate() API.
- a non-expect termination occurs making it susceptible to DoS attacks
- when the attacker could force interruptions of application processing,
- as the process terminates when accessing public key info of provided certificates from user code.
- The current context of users will then be gone.
- This vulnerability affected all active Node.js versions v16, v18, and v20.

CVE-2023-30588 What are X509 certificates?

- RFC 5280 (Request for Comments) defines X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.
- These certificates are used in many Internet protocols, including TLS/SSL, and they are also used in offline applications
- An X.509 certificate binds an identity (a hostname, or an organization, or an individual) to a public key using a digital signature,
- The X.509 certificate structure is defined using the ASN.1 (Abstract Syntax Notation One) standard, and describes rules and structures for representing, de/encoding, and transmitting (..).

CVE-2023-30588: What is Node.js?

- Node.js is a cross-platform, open-source JavaScript runtime environment that executes JavaScript code outside a web browser.
- It’s built on the V8 JavaScript engine and uses an event-driven, non-blocking I/O model, making it lightweight and efficient.
- This allows developers to use JavaScript for server-side scripting, to write command line tools, and for generating dynamic web page content before it’s sent to the user’s web browser.

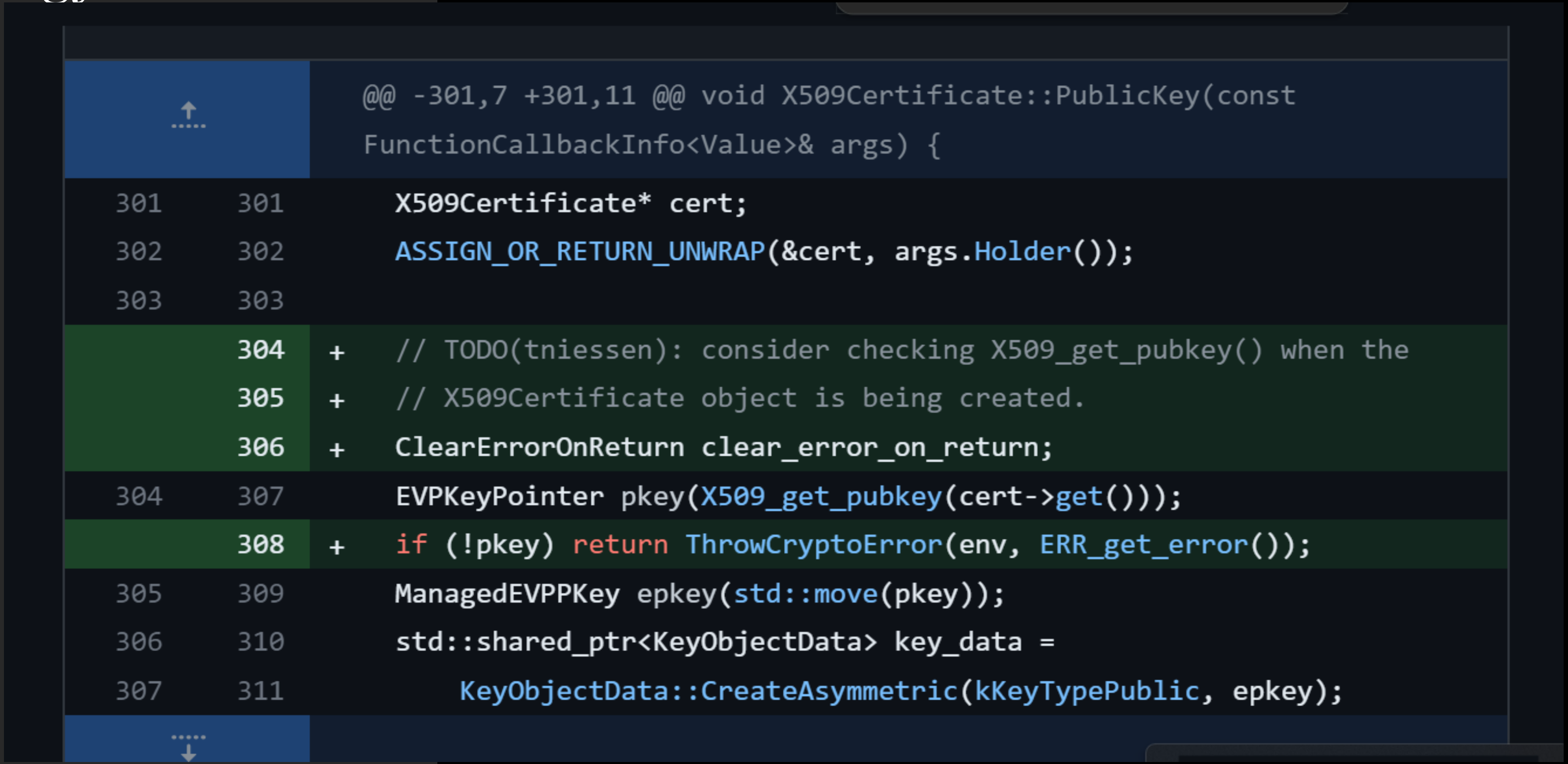
CVE-2023-30588: What is the x509Cert.. ctor good for?

- The crypto.X509Certificate(str) constructor in Node.js’s crypto module creates an instance of the X509Certificate class.
- The constructor takes a single argument, which is a buffer or string representing a PEM-encoded (Privacy Enhanced Mail) X.509 certificate.
- The X509Certificate instance provides methods to access information about the X.509 certificate, such as the subject, issuer, validity dates, and more.

CVE-2023-30588 What was the fix strategy?

- OpenSSL parsing of the x509 Certificate did not crash parsing the PoC certificate, because the file contains a structurally sound TBSCertificate (To be signed) structure.
- However, the SPKI (Simple PKI) field of the certificate contains the subjectPublicKey as an ASN.1 BIT STRING
- This bit sequence is not a valid public key, as assumed by the Node.js glue code to OpenSSL
- TL;DR: The fix is to add a check that the X509Certificate.publicKey function uses a valid public key and does not abort in this edge case

CVE-2023-30588 What was the fix strategy?



```
@@ -301,7 +301,11 @@ void X509Certificate::PublicKey(const FunctionCallbackInfo<Value>& args) {
    X509Certificate* cert;
    ASSIGN_OR_RETURN_UNWRAP(&cert, args.Holder());

+   // TODO(tniessen): consider checking X509_get_pubkey() when the
+   // X509Certificate object is being created.
+   ClearErrorOnReturn clear_error_on_return;
    EVPKeyPointer pkey(X509_get_pubkey(cert->get()));
+   if (!pkey) return ThrowCryptoError(env, ERR_get_error());
    ManagedEVPPKey epkey(std::move(pkey));
    std::shared_ptr<KeyObjectData> key_data =
        KeyObjectData::CreateAsymmetric(kKeyTypePublic, epkey);
```

CVE-2023-30588
In the debugger

```
$gdb --args node loadcert_poc.js

…

#5  0xf1 in node::Abort() ()
#6  0x..5e in node::Assert(..) ()
#7  0x..52 in node::crypto::KeyObjectData::CreateAsymmetric(..) ()
#8  0x..46 in node::crypto::X509Certificate::PublicKey(..) ()
#9  0x..f0 in v8::internal.. >(..) ()
#10 0x..2f in v8::internal::Builtin_HandleApiCall(..) ()
#11 0x..79 in Builtins_CEntry_Return1_DontSaveFPRegs.. ()
#12 0x..d0 in Builtins_InterpreterEntryTrampoline ()
```

CVE-2023-30588
The fuzzing harness

```
$ find openssl/fuzz/corpora/x509/ -type f | xargs -I III -t  node loadcert_poc_var.js III

node loadcert_poc_var.js openssl/fuzz/corpora/x509/c757bd1adb0e098ea74310bffe005eae2022ab7

v18.15.0
valid:Mar 17 11:00:02 2018 GMT
node[3602761]: ../src/crypto/crypto_keys.cc:869:static shared_ptr<KeyObjectData>
KeyObjectData::CreateAsymmetric(KeyType, const ManagedEVPPKey&): Assertion `pkey' failed.

 1: 0xb7b3e0 node::Abort()
 2: 0xb7b45e
 3: 0xd16c52 node::crypto::KeyObjectData::CreateAsymmetric(crypto::KeyType,
crypto::ManagedEVPPKey const&)
 4: 0xd2f246 node::crypto::X509Certificate::PublicKey(v8::FunctionCallbackInfo<v8::Value>
const&)
 5: 0xdc71f0
 6: 0xdc872f v8::internal::Builtin_HandleApiCall(int, unsigned long*, v8::internal::Isolate*)
 7: 0x1707c79

xargs: node: terminated by signal 6
```

TL; DR: Fuzzing strategy was to use an existing corpus, the first iteration failed, no further tries necessary.

CVE-2023-30588 Timeline

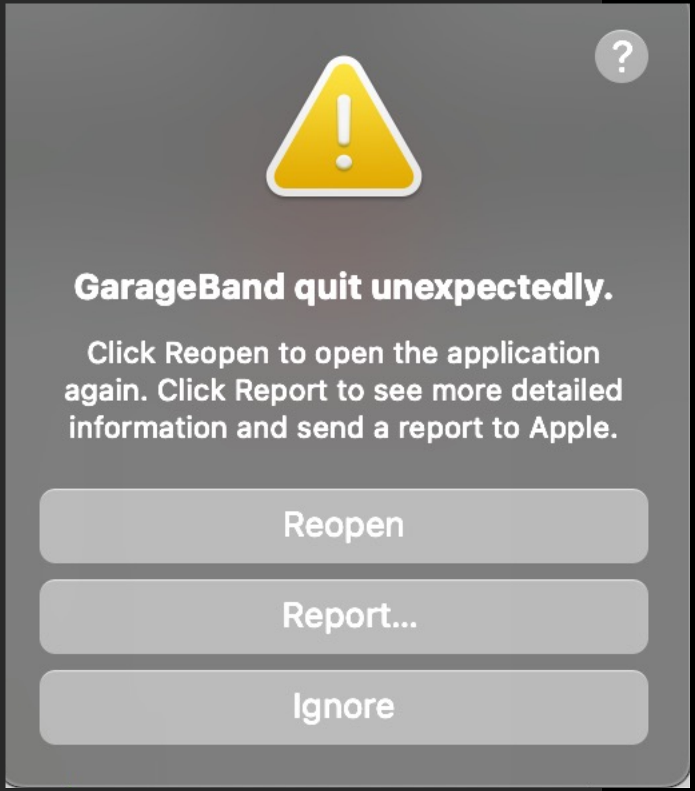
- Reported: February 23rd, 2023
- Confirmation: February 23rd, 2023
- Advisory: June 20th, 2023

Example 4: CVE-2024-23300

CVE-2024-23300

What was the bug?

- a use-after-free memory issue that could lead to “unexpected app termination or arbitrary code execution.”
- According to Forbes: “The former is annoying, but the latter could have substantial potential security issues should an attacker exploit this vulnerability. ”



GarageBand quit unexpectedly.

Click Reopen to open the application again. Click Report to see more detailed information and send a report to Apple.

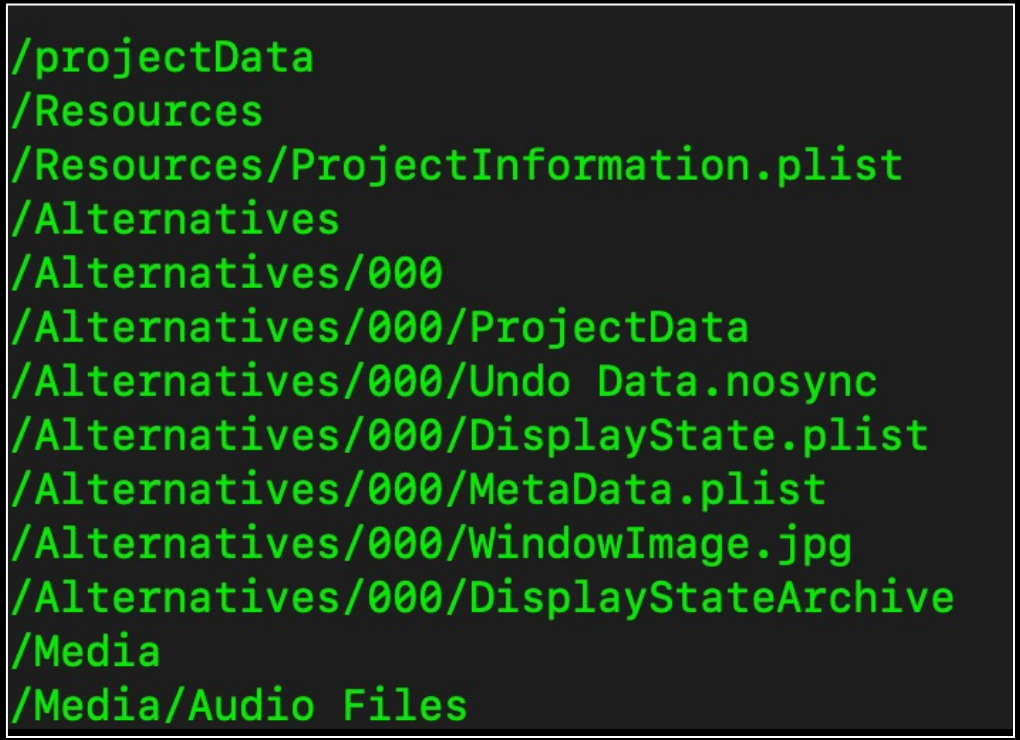
CVE-2024-23300

What are Garageband project files?

- GarageBand project files are directories (folders) that are treated for some purposes by the Mac OS as single files called a Bundle.
- GarageBand project files can be saved in the GarageBand subfolder
- located in the Music folder on your Mac computer, and they can also be easily.

CVE-2024-23300

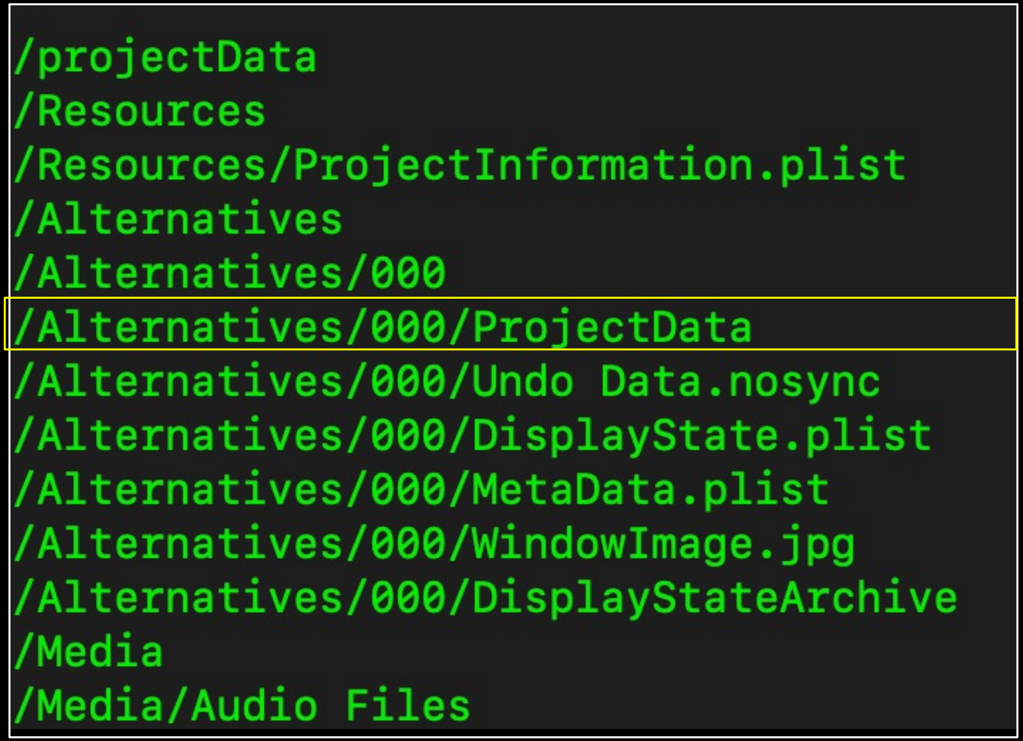
What are Garageband project files?



```
/projectData
/Resources
/Resources/ProjectInformation.plist
/Alternatives
/Alternatives/000
/Alternatives/000/ProjectData
/Alternatives/000/Undo Data.nosync
/Alternatives/000/DisplayState.plist
/Alternatives/000/MetaData.plist
/Alternatives/000/WindowImage.jpg
/Alternatives/000/DisplayStateArchive
/Media
/Media/Audio Files
```

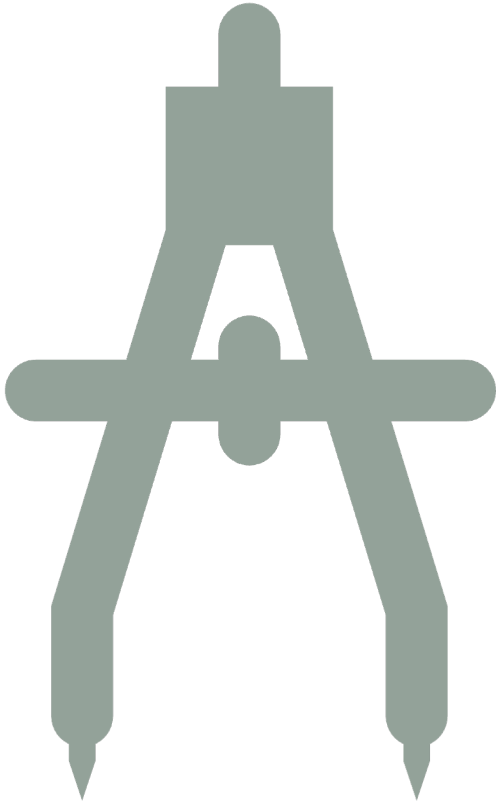
CVE-2024-23300

What are Garageband project files?



```
/projectData
/Resources
/Resources/ProjectInformation.plist
/Alternatives
/Alternatives/000
/Alternatives/000/ProjectData
/Alternatives/000/Undo Data.nosync
/Alternatives/000/DisplayState.plist
/Alternatives/000/MetaData.plist
/Alternatives/000/WindowImage.jpg
/Alternatives/000/DisplayStateArchive
/Media
/Media/Audio Files
```

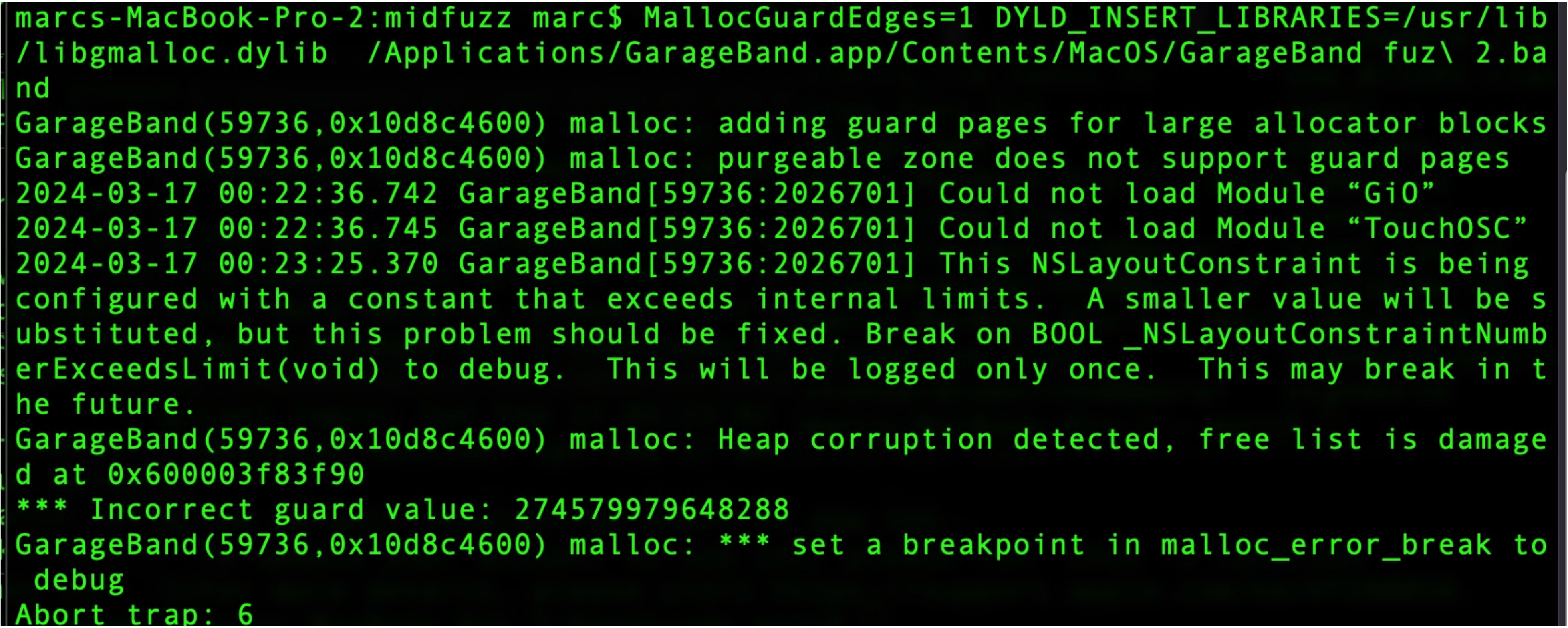
- Fuzzing candidate was the binary blob in ProjectData
- The other files are well-tested formats



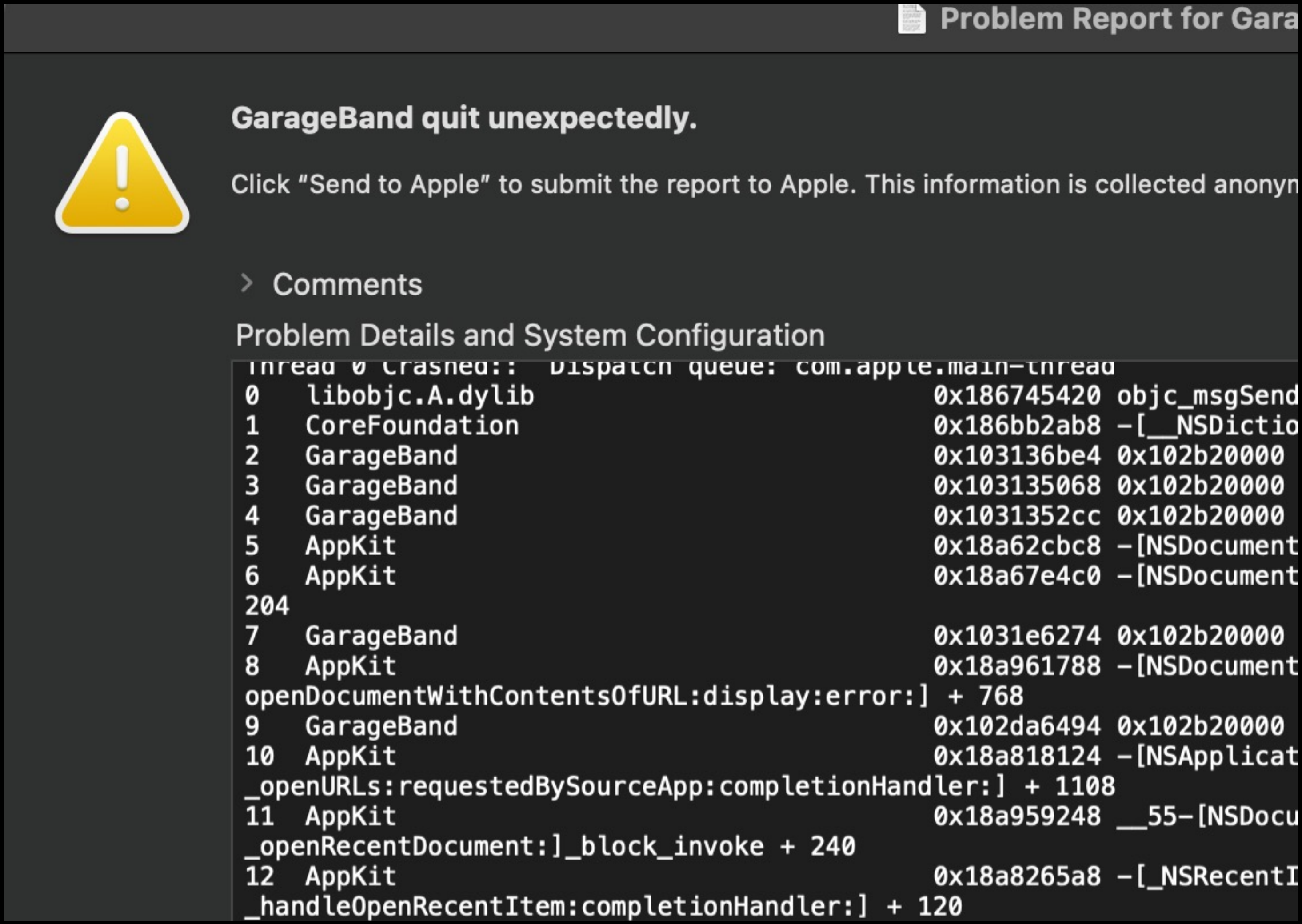
CVE-2024-23300
The low-tech fuzzing setup

- The approach was to go into GarageBand and click some random notes in the GUI
- Then saved the file
- Then the ProjectData file was repeatedly fuzzed via bit mutation (zzuf) and loaded into GarageBand
- In this endless loop waited for security-related crashes, which eventually happened

CVE-2024-23300
What was the fix strategy?



```
marcs-MacBook-Pro-2:midfuzz marc$ MallocGuardEdges=1 DYLD_INSERT_LIBRARIES=/usr/lib
/libgmalloc.dylib  /Applications/GarageBand.app/Contents/MacOS/GarageBand fuz\ 2.ba
nd
GarageBand(59736,0x10d8c4600) malloc: adding guard pages for large allocator blocks
GarageBand(59736,0x10d8c4600) malloc: purgeable zone does not support guard pages
2024-03-17 00:22:36.742 GarageBand[59736:2026701] Could not load Module “GiO”
2024-03-17 00:22:36.745 GarageBand[59736:2026701] Could not load Module “TouchOSC”
2024-03-17 00:23:25.370 GarageBand[59736:2026701] This NSLayoutConstraint is being
configured with a constant that exceeds internal limits.  A smaller value will be s
ubstituted, but this problem should be fixed. Break on BOOL _NSLayoutConstraintNumb
erExceedsLimit(void) to debug.  This will be logged only once.  This may break in t
he future.
GarageBand(59736,0x10d8c4600) malloc: Heap corruption detected, free list is damage
d at 0x600003f83f90
*** Incorrect guard value: 274579979648288
GarageBand(59736,0x10d8c4600) malloc: *** set a breakpoint in malloc_error_break to
 debug
Abort trap: 6
```



Problem Report for Gara

GarageBand quit unexpectedly.

Click “Send to Apple” to submit the report to Apple. This information is collected anonyn

Problem Details and System Configuration

```
Thread 0 Crashed::  Dispatch queue: com.apple.main-thread
0   libobjc.A.dylib                         0x186745420 objc_msgSend
1   CoreFoundation                          0x186bb2ab8 -[__NSDictio
2   GarageBand                              0x103136be4 0x102b20000
3   GarageBand                              0x103135068 0x102b20000
4   GarageBand                              0x1031352cc 0x102b20000
5   AppKit                                  0x18a62cbc8 -[NSDocument
6   AppKit                                  0x18a67e4c0 -[NSDocument
204
7   GarageBand                              0x1031e6274 0x102b20000
8   AppKit                                  0x18a961788 -[NSDocument
openDocumentWithContentsOfURL:display:error:] + 768
9   GarageBand                              0x102da6494 0x102b20000
10  AppKit                                  0x18a818124 -[NSApplicat
_openURLs:requestedBySourceApp:completionHandler:] + 1108
11  AppKit                                  0x18a959248 __55-[NSDocu
_openRecentDocument:]_block_invoke + 240
12  AppKit                                  0x18a8265a8 -[_NSRecentI
_handleOpenRecentItem:completionHandler:] + 120
```

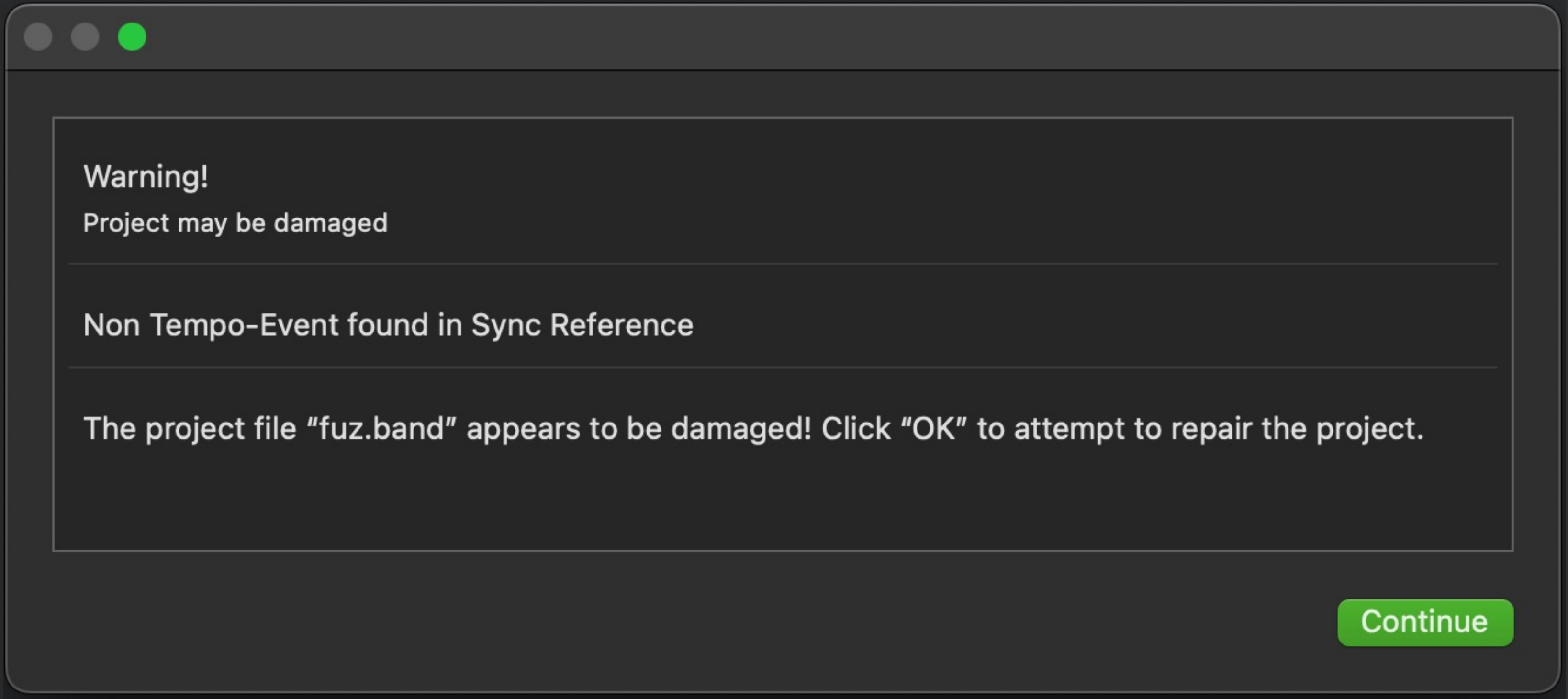
CVE-2024-23300 What was the fix strategy?

After detecting a crash the project state was restarted with GuardMalloc , which exposed the heap corruption.

The vulnerability is caused by a use-after-free condition, so the fix strategy was to “improve memory management”.

CVE-2024-23300

What was also annoying….



Warning!

Project may be damaged

Non Tempo-Event found in Sync Reference

The project file "fuz.band" appears to be damaged! Click "OK" to attempt to repair the project.

- Btw, where is the OK button, so I could "repair" the file?

CVE-2024-23300 Timeline

- Reported April 20, 2022
- Automated response April 20, 2022
- Time passed, and I forgot about it
- Tried again in 2024 with GB 10.4.8
  - still crashed
  - Sent a reminder on Feb 11, 2024
- Fixed
  - in GarageBand 10.4.11
  - on Mar 12, 2024
  - But not for Monterey-based MacPro (sigh)

A typical low-tech fuzzing harness

```
For $seed in $(seq 1 10 $max) ; do

    #Create PoC with $seed using zzuf , radamsa, afl-fuzz, honggfuzz
    # Run Poc , make sure you know all command line switches (implicit coverage!)
    # Monitor native memory handling with GuardMalloc, MALLOC_CHECK_, pageheap
    # cap execution time with the timeout command


If return code

    Save PoC , save crash info , update counters, ring bell

fi

done
```

Lessons learned

- The complexity of a fuzzer does not necessarily correlate with it‘s bug finding likelihood, as a simple approach may harvest interesting bugs
- A well documented fuzzing test plan may not always be an efficient test plan
- Low-tech fuzzing can be an essential technique to find bugs in high value software targets
- If successful for one software product, can additionally find bugs in dependent programs, especially in glue code
- Fuzzing corpora are a helpful vehicle to achieve sufficient coverage ahead-of-time , strategy should be good as long as we find bugs , reuse can be your friend to kickstart bug finding
- Starting with low tech fuzzing and later using advanced instrumented fuzzing are a great combo in a multi-step campaign workflow

Looking forward

- We likely just scratched the surface of discoverable bugs
- Keep on collecting and discovering fuzzing corpora and reapply it to potential consumers of these protocols
- Prioritize the blind spots in OSS-Fuzz fuzzing setups and go there (to what OSS-Fuzz does not exercise)
- Especially when low tech fuzzing has easily identified bugs, it seems promising to dig deeper with advanced fuzzing tools like AFL++
- TL;DR : Low tech fuzzing still has a place in the toolkit of security researchers to get a quick impression of the quality / stability of a product
- Therefore, expect more advisories as the ones presented here.

Q&A

(contact: https://de.linkedin.com/in/marcschoenefeld)

Finding more API use problems

Find candidates for inadequate use of OpenSSL API

```
apt-cache showpkg openssl (or rdepends)
Package: openssl
Versions:
3.0.2-0ubuntu1.15..

Reverse Depends:
  openssl-dbgsym,openssl 3.0.2-0ubuntu1.15
  lacme,openssl 1.1.0~
  python3-nova,openssl
```

- ..

Finding more API use problems

- … but the current Node.js package in Ubuntu does not appear in that list, because it uses the shared system library, it has an internal statically copy

-
  ```
  strings  /usr/lib/x86_64-linux-gnu/libnode.so.72  | grep OpenSSL | grep 20
  ```

-
  ```
  OpenSSL 1.1.1m  14 Dec 2021
  ```

- Fortunately, in the upstream LTS version via nvm (node version manager) has a current OpenSSL embedded

-
  ```
  $ strings –
  /home/user/.nvm/versions/node/v20.11.1/bin/node | grep OpenSSL | grep 202
  ```

-
  ```
  OpenSSL 3.0.13+quic 30 Jan 2024
  ```