Finding Error Handling Bugs in OpenSSL using Coccinelle

(Practical Experience Report)

Julia Lawall (University of Copenhagen/INRIA-Regal)

Ben Laurie (Google),
René Rydhof Hansen (Aalborg University),
Nicolas Palix (University of Copenhagen),
Gilles Muller (INRIA-Regal)

Context: Error handling in C code

The C language doesn't provide any error handling abstractions

- Convention 1: 1 indicates success, 0 indicates failure.
- Convention 2: 0 indicates success, -n indicates failure.

OpenSSL

- Toolkit for implementing secure network communication.
- Some OpenSSL functions return both 0 and -n on failure.

Problem 1

CVE-2008-5077 (January 2009)

- OpenSSL 0.9.8i and earlier does not properly check the return value from the EVP_VerifyFinal function, which allows remote attackers to bypass validation of the certificate chain via a malformed SSL/TLS signature for DSA and ECDSA keys.

Example:

```
if (!EVP_VerifyFinal(&md_ctx,p,(int)n,pkey)) {
  /* bad signature */
  al=SSL_AD_DECRYPT_ERROR;
  SSLerr(SSL_F_SSL3_GET_KEY_EXCHANGE,SSL_R_BAD_SIGNATURE);
  goto f_err;
}
```

But: EVP_VerifyFinal() returns 1 for a correct signature, 0 for failure and -1 if some other error occurred.

Problem 2

CVE-2009-0591 (March 2009)

- The CMS_verify function in OpenSSL 0.9.8h through 0.9.8j, when CMS is enabled, does not properly handle errors associated with malformed signed attributes, which allows remote attackers to repudiate a signature that originally appeared to be valid but was actually invalid.

Example:

```
if (!CMS_SignerInfo_verify_content(si, cmsbio)) {
  CMSerr(CMS_F_CMS_VERIFY, CMS_R_CONTENT_VERIFY_ERROR);
  goto err;
}
```

CMS_SignerInfo_verify_content() also returns 1, 0, or -1

Do similar bugs occur elsewhere?

Bugs in the CVE functions were fixed. Are there others?

Potential bug-finding methodology

- Find functions that return both 0 and negative values in error cases.
- Find uses of these functions that only test for 0.

Issues

- OpenSSL-1.0.0-stable-SNAP-20090911 contains almost 250 000 lines of C code and almost 6000 functions.
- Potentially many such functions and call sites, so automation is needed.

Our technology: Coccinelle

Features:

- Code-like notation for expressing searches.
- Patch features for expressing transformations (Semantic Patches).
- *Isomorphisms* for handling syntactic variations.

```
@@ expression list args; @@
- !EVP_VerifyFinal(args)
+ EVP_VerifyFinal(args) <= 0

@@ expression list args; @@
- !CMS_SignerInfo_verify_content(args)
+ CMS_SignerInfo_verify_content(args) <= 0
```

Problem: All function names must be known.

An iterative process, developed for Linux [DSN 2009]

Define a semantic patch to find functions having some property

Define a semantic patch template to find bugs in the use of an arbitrary function

Then, apply these semantic patches to the code base

An iterative process, developed for Linux [DSN 2009]

Define a semantic patch to find functions having some property

- A function that returns a negative constant directly.
- A function that stores a negative constant in a variable and returns that variable.
- A function that checks that a variable is negative and returns that variable.

Define a semantic patch template to find bugs in the use of an arbitrary function

Then, apply these semantic patches to the code base

An iterative process, developed for Linux [DSN 2009]

Define a semantic patch to find functions having some property

Define a semantic patch template to find bugs in the use of an arbitrary function

- Code that only checks whether the result is 0, not whether it is negative.

Then, apply these semantic patches to the code base

An iterative process, developed for Linux [DSN 2009]

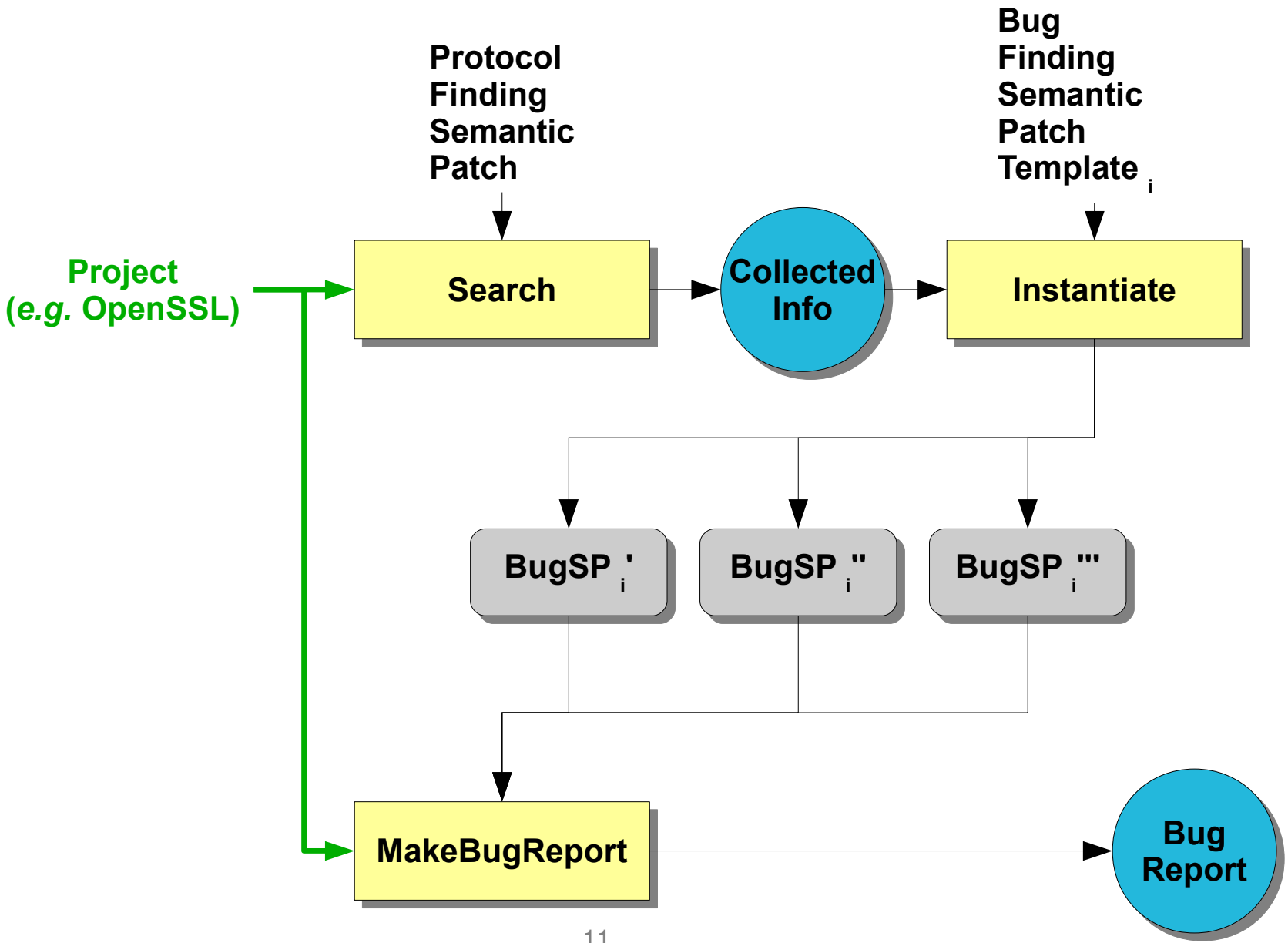
Define a semantic patch to find functions having some property

Define a semantic patch template to find bugs in the use of an arbitrary function

Then, apply these semantic patches to the code base

- Run the first semantic patch on the code base to collect a list of function names.
- Instantiate the semantic patch template for each collected function.
- Run the instantiated semantic patches on the code base to find and fix the bugs.

An iterative process, developed for Linux [DSN 2009]



Results

387 functions of the three types identified

```
@@
expression list args;
identifier virtual.FN;
@@
-    (FN(args)) == 0
+    FN(args) <= 0

@@
expression list args;
identifier virtual.FN;
@@
-    (FN(args)) != 0
+    FN(args) > 0
```

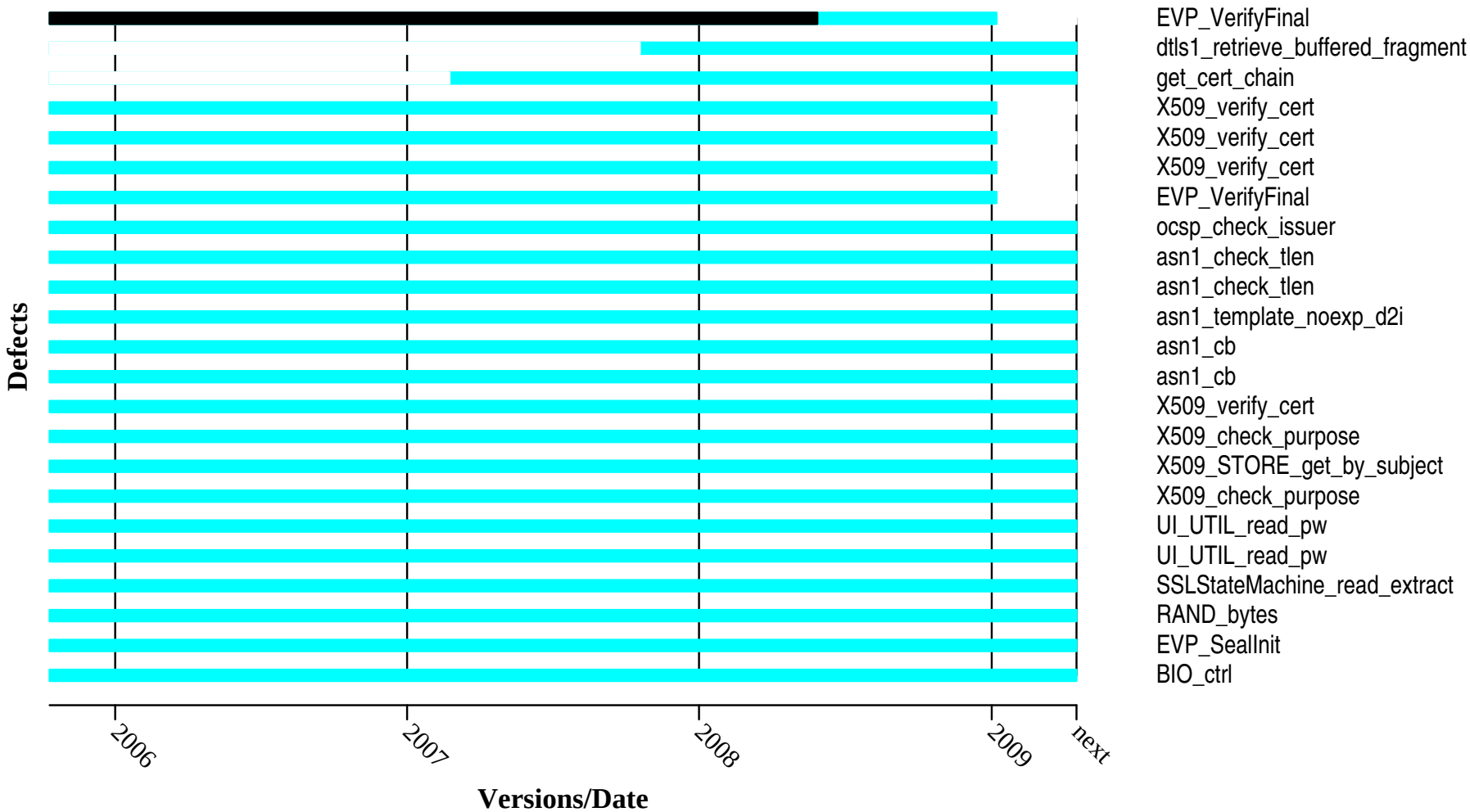
Bugs: 26
False positives: 20
Unknown: 3
Files: 30

```
@match@
expression x, E; constant C;
identifier virtual.FN;
position p;
@@

x@p = FN(...)
<+... when != x <= ( 0 | -C )
      when != x <  ( 0 | -C )
      when != ( x > 0 | x == -C )
 ( x != 0 | x == 0 )
...+>
( return ...; | x = E )
```

Bugs: 6
False positives: 14
Unknown: 2
Files: 19

Bug history


Defects
2006
2007
2008
2009
next
Versions/Date
EVP_VerifyFinal
dtls1_retrieve_buffered_fragment
get_cert_chain
X509_verify_cert
X509_verify_cert
X509_verify_cert
EVP_VerifyFinal
ocsp_check_issuer
asn1_check_tlen
asn1_check_tlen
asn1_template_noexp_d2i
asn1_cb
asn1_cb
X509_verify_cert
X509_check_purpose
X509_STORE_get_by_subject
X509_check_purpose
UI_UTIL_read_pw
UI_UTIL_read_pw
SSLStateMachine_read_extract
RAND_bytes
EVP_SealInit
BIO_ctrl

Issues

OpenSSL-specific macros

- `STACK_OF(SSL_COMP) *sk;` is not valid C.
- Solution: Configure Coccinelle to ignore `STACK_OF` (4 problematic macros in all)

Functions using 0 for success

- Some functions return 0 for success and negative values for failure.
- Solution: Filter out functions that never return positive values.

Issues

Comparison functions

- Some functions return -1 for <, 0 for =, and 1 for >
- Solution: Filter out function names ending in `cmp`.

Value dependencies

- Function arguments may control whether a negative result is possible.

```
if ((a != NULL) && (sk_num(a) != 0)) M_ASN1_I2D_put_SET(a,f);
```

- Solution: Manual inspection.
- Potential solution: Data flow analysis (integration with Clang).

Conclusions

Our technique that was developed for Linux code has been shown useful for (user-level) OpenSSL code too.

Different projects have different conventions, bug histories and bug profiles.

- Our previous efforts with OpenSSL found few bugs, and those found did not interest the OpenSSL community.
- Bug-finding requires project-specific expertise.
- Automated bug-finding tools must be easily adaptable by the user.

Coccinelle can meet this challenge.

http://coccinelle.lip6.fr/