where does bug come from 🐛

Yu Shen

2017.6

experiences

- 2 years
- 200+ bug fixs

bugs can be deadly

```
if (launch = true)
{
    launch_missile();
}
```

bugs can be deadly

bugs can be deadly

a shell script to clean the TMPDIR.

```
rm -rf $TMPDIR/
```

bugs can be deadly

a script to clean the TMPDIR.

```
rm -rf $TMPDIR/*
```

but when `$TMPDIR` is empty 😢

if you see

- core dump
- freeze
- unexpected error message
- software behave in unintended ways

There is likey to be a bug...

all kinds of bugs

- bad programming practice
  - dangerous C functions: strcmp, strcpy, strcat, sprintf ...
  - function is not declared before use
  - forgot to check input arguments
- memory management
  - buffers overflow
  - free memory on stack
  - de-referencing/accessing NULL pointers
  - memory leak

bugs in catogories

- concurrency
  - without locks
  - dead locks
  - TLS - thread local storage
- compiler optimization
- buggy new features/codes
- bad design

bad programming practice

example1: strcmp

strcmp

```
if (strcmp(te->desc, "MATERIALIZED VIEW DATA"))
{
...
```

click to see on gerrit

strcmp

```
if (strcmp(te->desc, "MATERIALIZED VIEW DATA") == 0)
{
...
```

click to see on gerrit

bad programming practice

example2: function is not declared before use

function is not declared before use

```
#0 strlen ()
#1 vfprintf ()
#2 vsnprintf ()
#3 appendStringInfoVA (str=0x7fffb327c000,
   fmt=0x3e87260 "table %s is remapped to %s by mapping rule \"
#4 elog_finish (elevel=13, fmt=0x4f9de0 "table %s is remapped t
#5 ApplyCatalogMappingRules
```

click to see on gerrit

function is not declared before use

```
char *
ApplyCatalogMappingRules(
        const char *schema, const char *object)
{
  rewritten_fqname = filter_RE_replace(fqname, li_entry->re,  l
  elog(DEBUG2, "table %s is remapped to %s by mapping rule \"%s
```

```
(gdb) p rewritten_fqname
$2 = 0xffffffffd53b9d90
    <Address 0xffffffffd53b9d90 out of bounds>
```

click to see on gerrit

function is not declared before use

```
(gdb) disas filter_RE_replace
...
<filter_RE_replace+214>:    callq  0x45a54a <text_to_cstring>
<filter_RE_replace+219>:    cltq
...
```

click to see on gerrit

function is not declared before use

if the function is not declared before use. the complier will assume the return type of the function to be int32 which is not enough to store 64bit memory address. and later use 'cltq' to extend the int32(eax) to int64(rax) which is an illegal memory address.

to fix the bug, add declaration before use.

```
extern char *text_to_cstring(const text *t);
```

click to see on gerrit

bad programming practice

example3: forget to check input arguments

forgot to check input arguments

```
sword
DCIBindArrayOfStruct(DCIBind *bindp, DCIError *errhp,
ub4 pvskip, ub4 indskip,
ub4 alskip, ub4 rcskip)
{
        mylog("[DCIBindArrayOfStruct]: bindp = %p, pvskip = %u,
        bindp->pvskip = pvskip;
        bindp->indskip = indskip;
...
```

forgot to check input arguments

```
sword
DCIBindArrayOfStruct(DCIBind *bindp, DCIError *errhp,
                                        ub4 pvskip, ub4 indski
                                        ub4 alskip, ub4 rcskip
{
    if (NULL == bindp ||
    bindp->head.handle_type != DCI_HTYPE_BIND)
        return DCI_INVALID_HANDLE;
...
```

memory management

example1: buffers overflow

buffers overflow

```
strcpy(NameStr(*change->llogdata->schema), schema_name);
strcpy(NameStr(*change->llogdata->object), object_name);
```

click to see on gerrit

buffers overflow

use `strncpy` instead of `strcpy`

```
strncpy(NameStr(*change->llogdata->schema),
        schema_name,
        NAMEDATALEN);
strncpy(NameStr(*change->llogdata->object),
        object_name,
        NAMEDATALEN);
```

click to see on gerrit

memory management

example2: free memory on stack

free memory on stack

```
char
QR_read_tuple(QResultClass *self, char binary)
{
  char  tidoidbuf[32];
  if (field_lf >= effective_cols)
    buffer = tidoidbuf;
  else
    buffer = (char *) malloc(len + 1);
...
this_tuplefield[field_lf].value = buffer;
...
```

```
void
QR_free_memory(QResultClass *self)
{
        free(tuple[lf].value);
...
```

click to see on gerrit

memory management

example3: accessing NULL pointers

accessing NULL pointers

```
else if(errornum == SOCKET_CLOSED)
{
    DBC_set_fullerror(self,
    HYT00_SOCKET_NOTEXPECT_ERROR,
    sock->errormsg,
    "08S01");
```

click to see on gerrit

accessing NULL pointers

```
#define SOCK_get_errmsg(self) \
        (self ? self->errormsg : "socket closed")
...
else if(errornum == SOCKET_CLOSED)
{
        DBC_set_fullerror(self,
        HYT00_SOCKET_NOTEXPECT_ERROR,
        SOCK_get_errmsg(sock),
        "08S01");
```

click to see on gerrit

memory management

example4: memory leak

memory leak

```
HeapTuple
BuildTupleFromCStrings(AttInMetadata *attinmeta, char **values)
{
...
  for (i = 0; i < natts; i++)
  {
     dvalues[i] =
     InputFunctionCall(&attinmeta->attinfuncs[i],
                       values[i],
...
tuple = heap_formtuple(tupdesc, dvalues, nulls);
...
return tuple;
```

click to see on gerrit

memory leak

```
for (i = 0; i < natts; i++)
{
 /*
  * Free the mem allocated in xxx_in to avoid memory leak
  */
  switch(tupdesc->attrs[i]->atttypid)
  {
  /* Type below are pass by ref, e.g. see numeric_in */
  case NUMERICOID:
  case VARCHAROID:
  case TEXTOID:
  case INT2VECTOROID:
  case TIDOID:
  case OIDVECTOROID:
    if (NULL != dvalues[i])
    {
      pfree((void *)(dvalues[i]));
    }
    break;
```

click to see on gerrit

TLS - thread local storage

```
ODBCEnv odbcEnvHandle;
```

click to see on gerrit

TLS - thread local storage

```
MT_LOCAL ODBCEnv odbcEnvHandle = NULL;
```

click to see on gerrit

compiler optimization

compiler optimization

```
UCHAR
SOCK_get_next_byte(SocketClass *self)
{
self->buffer_filled_in = recv(self->socket, (char *) self->buff
if (self->buffer_filled_in < 0)
{
...
SocketClass *sc_temp = self; /* remember the pointer value. */
readycode = SOCK_wait_for_ready(self, FALSE, retry_count);
if (NULL == self)
{
  if (sc_temp != NULL)
    self = sc_temp;
...
```

compiler optimization

To avoid compiler optimization:

1. we avoid to use

```
if (NULL == self)
{
  ... code here will be optimized by gcc -O2
  because SOCK_get_next_byte check self != NULL
  if reach here gcc think self must not be NULL
}
```

2. we have to make self_value and self volatile

compiler optimization

```
volatile void * self_value;
/*
* The below C code in asm looks like:
*           ...
*                        mov    %rbx,0x8(%rsp) --save self
*                        callq  595d0 <SOCK_wait_for_ready>
*                        test   %eax,%eax
*                        mov    0x8(%rsp),%rbx --restore self
*           ...
* if you change the code you have to check the asm
* that the self is saved and restored.
*/
self_value = (void *)self; /* remember the pointer value. */
readycode = SOCK_wait_for_ready(self, FALSE, retry_count);
self = (SocketClass *)self_value; /* restore self */
```

click to see on gerrit

buggy new features/codes

buggy new features/codes

dump partation table

dump partation table

```
for (cell = patterns->head; cell; cell = cell->next)
{
  isparttab = parsePartition(cell->val, &maintab, &parttab);
  /* bug24408 add _PRT_oid_ to parttab */
  if (isparttab)
    partitionNameAddOid(maintab, &parttab);
```

When add new features, do not forget dump/restore, replication...

how to avoid bugs

- think twice before you type
- 40% of coments/documentation.
- don't ignore compiler warnings
- defensive programming: do not coredump in my code!
- use tools such as lint/valgrind to find out possible bugs
- unit test and regression test: code coverage

Thanks!

Follow me on https://www.shenyu.wiki