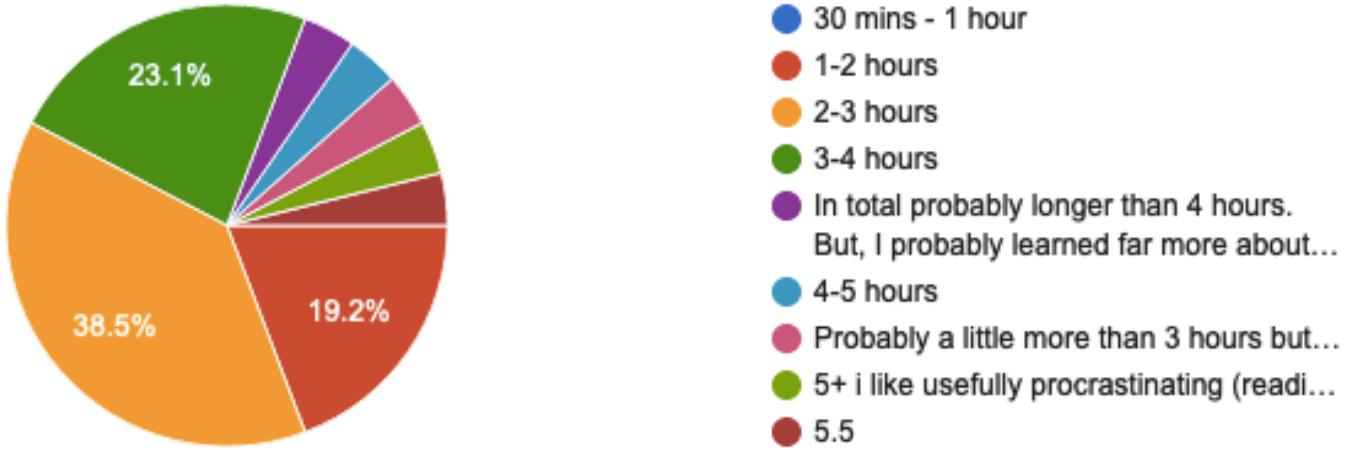
Ownership (cont.) and Error Handling

Ryan Eberhardt and Armin Namavari
April 14, 2020

Congrats on finishing week 1!

How much time did you spend on the week 1 exercises?

26 responses




frustrating
Rust
assignment
learning
helpful
definitely
really
like
Im
syntax
time
everything
understand
understanding
didnt
first
simple
wasnt
trivial
hacky
versus
nothing
learned
better
class
felt
beginning
type
start
pick
pretty
good
just
going
getting
curve
strings
error
much
since
code
still
things
think
overall
especially
hard
likely
starting
work
get
lot
far
come
fun
well
make
kind
part
instructions
hangman
mut
got
types
two

General notes

- If you ever need an extension, just let us know
  - This class is supposed to be fun
  - Sleep deprivation -> coronavirus
- This class is in Rust, but it's not a Rust class
  - You Won't Believe This One Weird Fact
  - This class is more about exposure to ideas you can take with you
  - Rust is a response to the problems of C/C++. If you never use Rust again in your life, it would still be good to know about
    - The problems with C/C++
    - How people are responding
    - The problems with that response
  - There are lots of great questions on Slack. Don't be intimidated by fancy lingo flying around

Today’s lecture

- Recap ownership
- Work through some examples of ownership in code
- Talk about error handling in Rust

Ownership

Ownership — in C!

```
/* Get status of the virtual port (ex. tunnel, patch).
 *
 * Returns '0' if 'port' is not a virtual port or has no errors.
 * Otherwise, stores the error string in '*errp' and returns positive errno
 * value. The caller is responsible for freeing '*errp' (with free()).
 *
 * This function may be a null pointer if the ofproto implementation does
 * not support any virtual ports or their states.
 */
int (*vport_get_status)(const struct ofport *port, char **errp);
```

Open vSwitch

```c
/**
 * @note Any old dictionary present is discarded and replaced with a copy of the new one. The
 * caller still owns val is and responsible for freeing it.
 */
int av_opt_set_dict_val(void *obj, const char *name, const AVDictionary *val, int search_flags);
```

ffmpeg

```c
/**
 * iscsi_boot_create_target() - create boot target sysfs dir
 * @boot_kset: boot kset
 * @index: the target id
 * @data: driver specific data for target
 * @show: attr show function
 * @is_visible: attr visibility function
 * @release: release function
 *
 * Note: The boot sysfs lib will free the data passed in for the caller
 * when all refs to the target kobject have been released.
 */
struct iscsi_boot_kobj *
iscsi_boot_create_target(struct iscsi_boot_kset *boot_kset, int index,
                         void *data,
                         ssize_t (*show) (void *data, int type, char *buf),
                         umode_t (*is_visible) (void *data, int type),
                         void (*release) (void *data))
{
        return iscsi_boot_create_kobj(boot_kset, &iscsi_boot_target_attr_group,
                                      "target%d", index, data, show, is_visible,
                                      release);
}
EXPORT_SYMBOL_GPL(iscsi_boot_create_target);
```

Linux kernel

```
/* Looks up a port named 'devname' in 'ofproto'.  On success, returns 0 and
 * initializes '*port' appropriately. Otherwise, returns a positive errno
 * value.
 *
 * The caller owns the data in 'port' and must free it with
 * ofproto_port_destroy() when it is no longer needed. */
int (*port_query_by_name)(const struct ofproto *ofproto,
                          const char *devname, struct ofproto_port *port);
```

Open vSwitch

```
/**
 * dvb_unregister_frontend() - Unregisters a DVB frontend
 *
 * @fe: pointer to &struct dvb_frontend
 *
 * Stops the frontend kthread, calls dvb_unregister_device() and frees the
 * private frontend data allocated by dvb_register_frontend().
 *
 * NOTE: This function doesn't frees the memory allocated by the demod,
 * by the SEC driver and by the tuner. In order to free it, an explicit call to
 * dvb_frontend_detach() is needed, after calling this function.
 */
int dvb_unregister_frontend(struct dvb_frontend *fe);
```

Linux kernel

```
static void mapper_count_similar_free(mapper_t* pmapper, context_t* _) {
    mapper_count_similar_state_t* pstate = pmapper->pvstate;
    slls_free(pstate->pgroup_by_field_names);

    // lhmslv_free will free the keys: we only need to free the void-star values.
    for (lhmslve_t* pa = pstate->pcounts_by_group->phead; pa != NULL; pa = pa->pnext) {
        unsigned long long* pcount = pa->pvvalue;
        free(pcount);
    }
    lhmslv_free(pstate->pcounts_by_group);

    ...
}
```

Miller

Compile time vs run time

What does my Rust code actually do?

- Passing ownership: just passes a pointer
  - The compiler will insert the appropriate free() call for you
- Passing references: just passes a pointer
- Explicit copy: copies memory!

Will it compile?

Live demo

“One thing that’s confusing is why sometimes I need to &var and other times I can just use var: for example, set.contains(&var), but set.insert(var) – why?"

Error handling

```
// Imagine this is code for a network server that has just received and is
// processing a packet of data.
size_t len = packet.length;
void *buf = malloc(len);
memcpy(buf, packet.data, len);
// Do stuff with buf
// ...
free(buf);
```

Two issues

- Use of NULL in place of a real value
- Lack of proper error handling

Handling nulls

*“**I call it my billion-dollar mistake.** It was the invention of the null reference in 1965. At that time, I was designing the first comprehensive type system for references in an object oriented language (ALGOL W). My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.”*

- Tony Hoare

Common Vulnerabilities and Exposures

CVE List▾ CNAs▾ WGs▾ Board▾ About▾ News & Blog▾

NVD
Go to for:
CVSS Scores
CPE Info
Advanced Search

Search CVE List | Download CVE | Data Feeds | Request CVE IDs | Update a CVE Entry

TOTAL CVE Entries: 133847

HOME > CVE > SEARCH RESULTS

Search Results

There are **1627** CVE entries that match your search.

Name	Description
CVE-2020-9759	An issue was discovered in WeeChat before 2.7.1 (0.4.0 to 2.7 are affected). A malformed message 352 (who) can cause a NULL pointer dereference in the callback function, resulting in a crash.
CVE-2020-9385	A NULL Pointer Dereference exists in libzint in Zint 2.7.1 because multiple + characters are mishandled in add_on in upcean.c, when called from eanx in upcean.c during EAN barcode generation.
CVE-2020-9327	In SQLite 3.31.1, isAuxiliaryVtabOperator allows attackers to trigger a NULL pointer dereference and segmentation fault because of generated column optimizations.
CVE-2020-8859	This vulnerability allows remote attackers to create a denial-of-service condition on affected installations of ELOG Electronic Logbook 3.1.4-283534d. Authentication is not required to exploit this vulnerability. The specific flaw exists within the processing of HTTP parameters. A crafted request can trigger the dereference of a null pointer. An attacker can leverage this vulnerability to create a denial-of-service condition. Was ZDI-CAN-10115.
CVE-2020-8448	In OSSEC-HIDS 2.7 through 3.5.0, the server component responsible for log analysis (ossec-analysisd) is vulnerable to a denial of service (NULL pointer dereference) via crafted messages written directly to the analysisd UNIX domain socket by a local user.
CVE-2020-8011	CA Unified Infrastructure Management (Nimsoft/UIM) 9.20 and below contains a null pointer dereference vulnerability in the robot (controller) component. A remote attacker can crash the Controller service.
CVE-2020-8002	A NULL pointer dereference in vrend_renderer.c in virglrenderer through 0.8.1 allows attackers to cause a denial of service via commands that attempt to launch a grid without previously providing a Compute Shader (CS).
CVE-2020-7105	async.c and dict.c in libhiredis.a in hiredis through 0.14.0 allow a NULL pointer dereference because malloc return values are unchecked.
CVE-2020-7062	In PHP versions 7.2.x below 7.2.28, 7.3.x below 7.3.15 and 7.4.x below 7.4.3, when using file upload functionality, if upload progress tracking is enabled, but session.upload_progress.cleanup is set to 0 (disabled), and the file upload fails, the upload procedure would try to clean up data that does not exist and encounter null pointer dereference, which would likely lead to a crash.
CVE-2020-6795	When processing a message that contains multiple S/MIME signatures, a bug in the MIME processing code caused a null pointer dereference, leading to an unexploitable crash. This vulnerability affects Thunderbird < 68.5.
CVE-2020-6631	An issue was discovered in GPAC version 0.8.0. There is a NULL pointer dereference in the function gf_m2ts_stream_process_pmt() in media_tools/m2ts_mux.c.

NULL pointer dereferences

Why are NULLs so dangerous?

What should we do about it?

```rust
fn feeling_lucky() -> Option<String> {
    if get_random_num() > 10 {
        Some(String::from("I'm feeling lucky!"))
    } else {
        None
    }
}
```

```rust
fn feeling_lucky() -> Option<String> {
    if get_random_num() > 10 {
        Some(String::from("I'm feeling lucky!"))
    } else {
        None
    }
}
```

```rust
if feeling_lucky().is_none() {
    println!("Not feeling lucky :(");
}
```

```rust
fn feeling_lucky() -> Option<String> {
    if get_random_num() > 10 {
        Some(String::from("I'm feeling lucky!"))
    } else {
        None
    }
}
```

```rust
let message = feeling_lucky().unwrap_or(String::from("Not lucky :("));
```

```rust
fn feeling_lucky() -> Option<String> {
    if get_random_num() > 10 {
        Some(String::from("I'm feeling lucky!"))
    } else {
        None
    }
}
```

```rust
match feeling_lucky() {
    Some(message) => {
        println!("Got message: {}", message);
    },
    None => {
        println!("No message returned :-/");
    },
}
```

Handling errors

Error handling in C

- If a function might encounter an error, its return type is made to be `int` (or sometimes `void*`).
- If the function is successful, it returns `0`. Otherwise, if an error is encountered, it returns `-1`. (If the function is returning a pointer, it returns a valid pointer in the success case, or `NULL` if an error occurs.)
- The function that encountered the error sets the global variable `errno` to be an integer indicating what went wrong. If the caller sees that the function returned `-1` or `NULL`, it can check `errno` to see what error was encountered

```
#define EPERM            1      /* Operation not permitted */
#define ENOENT           2      /* No such file or directory */
#define ESRCH            3      /* No such process */
#define EINTR            4      /* Interrupted system call */
#define EIO              5      /* I/O error */
#define ENXIO            6      /* No such device or address */
#define E2BIG            7      /* Arg list too long */
#define ENOEXEC          8      /* Exec format error */
#define EBADF            9      /* Bad file number */
#define ECHILD          10      /* No child processes */
#define EAGAIN          11      /* Try again */
#define ENOMEM          12      /* Out of memory */
#define EACCES          13      /* Permission denied */
#define EFAULT          14      /* Bad address */
#define ENOTBLK         15      /* Block device required */
#define EBUSY           16      /* Device or resource busy */
#define EEXIST          17      /* File exists */
#define EXDEV           18      /* Cross-device link */
#define ENODEV          19      /* No such device */
#define ENOTDIR         20      /* Not a directory */
#define EISDIR          21      /* Is a directory */
#define EINVAL          22      /* Invalid argument */
#define ENFILE          23      /* File table overflow */
#define EMFILE          24      /* Too many open files */
#define ENOTTY          25      /* Not a typewriter */
#define ETXTBSY         26      /* Text file busy */
#define EFBIG           27      /* File too large */
#define ENOSPC          28      /* No space left on device */
#define ESPIPE          29      /* Illegal seek */
#define EROFS           30      /* Read-only file system */
#define EMLINK          31      /* Too many links */
#define EPIPE           32      /* Broken pipe */
#define EDOM            33      /* Math argument out of domain of func */
#define ERANGE          34      /* Math result not representable */
#define EDEADLK         35      /* Resource deadlock would occur */
#define ENAMETOOLONG    36      /* File name too long */
#define ENOLCK          37      /* No record locks available */
#define ENOSYS          38      /* Function not implemented */
#define ENOTEMPTY       39      /* Directory not empty */
#define ELOOP           40      /* Too many symbolic links encountered */
#define EWOULDBLOCK     EAGAIN  /* Operation would block */
#define ENOMSG          42      /* No message of desired type */
#define EIDRM           43      /* Identifier removed */
#define ECHRNG          44      /* Channel number out of range */
#define EL2NSYNC        45      /* Level 2 not synchronized */
#define EL3HLT          46      /* Level 3 halted */
#define EL3RST          47      /* Level 3 reset */
#define ELNRNG          48      /* Link number out of range */
#define EUNATCH         49      /* Protocol driver not attached */
#define ENOCSI          50      /* No CSI structure available */
#define EL2HLT          51      /* Level 2 halted */
#define EBADE           52      /* Invalid exchange */
#define EBADR           53      /* Invalid request descriptor */
#define EXFULL          54      /* Exchange full */
#define ENOANO          55      /* No anode */
#define EBADRQC         56      /* Invalid request code */
#define EBADSLT         57      /* Invalid slot */
#define EBFONT          59      /* Bad font file format */
#define ENOSTR          60      /* Device not a stream */
#define ENODATA         61      /* No data available */
#define ETIME           62      /* Timer expired */
#define ENOSR           63      /* Out of streams resources */
#define ENONET          64      /* Machine is not on the network */
#define ENOPKG          65      /* Package not installed */
#define EREMOTE         66      /* Object is remote */
#define ENOLINK         67      /* Link has been severed */
#define EADV            68      /* Advertise error */
#define ESRMNT          69      /* Srmount error */
#define ECOMM           70      /* Communication error on send */
#define EPROTO          71      /* Protocol error */
#define EMULTIHOP       72      /* Multihop attempted */
#define EDOTDOT         73      /* RFS specific error */
#define EBADMSG         74      /* Not a data message */
#define EOVERFLOW       75      /* Value too large for defined data type */
#define ENOTUNIQ        76      /* Name not unique on network */
#define EBADFD          77      /* File descriptor in bad state */
#define EREMCHG         78      /* Remote address changed */
#define ELIBACC         79      /* Can not access a needed shared library */
#define ELIBBAD         80      /* Accessing a corrupted shared library */
#define ELIBSCN         81      /* .lib section in a.out corrupted */
#define ELIBMAX         82      /* Attempting to link in too many shared libraries */
#define ELIBEXEC        83      /* Cannot exec a shared library directly */
#define EILSEQ          84      /* Illegal byte sequence */
#define ERESTART        85      /* Interrupted system call should be restarted */
#define ESTRPIPE        86      /* Streams pipe error */
#define EUSERS          87      /* Too many users */
#define ENOTSOCK        88      /* Socket operation on non-socket */
#define EDESTADDRREQ    89      /* Destination address required */
#define EMSGSIZE        90      /* Message too long */
#define EPROTOTYPE      91      /* Protocol wrong type for socket */
#define ENOPROTOOPT     92      /* Protocol not available */
#define EPROTONOSUPPORT 93      /* Protocol not supported */
#define ESOCKTNOSUPPORT 94      /* Socket type not supported */
#define EOPNOTSUPP      95      /* Operation not supported on transport endpoint */
#define EPFNOSUPPORT    96      /* Protocol family not supported */
#define EAFNOSUPPORT    97      /* Address family not supported by protocol */
#define EADDRINUSE      98      /* Address already in use */
#define EADDRNOTAVAIL   99      /* Cannot assign requested address */
...
```

```
ssize_t siz = msgrcv(msqid, msgp, msgsz, msgtyp, msgflg);
if (siz<0) { // msgrcv failed and has set errno
  if (errno == ENOMSG)
    dosomething();
  else if (errno == EAGAIN)
    dosomethingelse();
  /// etc
  else {
     syslog(LOG_DAEMON|LOG_ERR, "msgrcv failure with %s\n",
            strerror(errno));
     exit(EXIT_FAILURE);
  };
};
```

https://stackoverflow.com/questions/46013418/how-to-check-the-value-of-errno

CVE-2015-8812

- Critical Linux kernel vulnerability: by sending a malformed network packet, a remote attacker could execute arbitrary code in the kernel
- A set of kernel networking functions were returning -1 for error, 0 for success, but also other values for “warnings”
  - Returned `NET_XMIT_CN` (defined to be 2) when congestion was detected
- Code calling these functions saw nonzero return code and assumed there was a network error
- Freed memory that was still being used for the network. Use-after-free + double free!

The fix

```
--- a/drivers/infiniband/hw/cxgb3/iwch_cm.c
+++ b/drivers/infiniband/hw/cxgb3/iwch_cm.c
@@ -149,7 +149,7 @@ static int iwch_l2t_send(struct t3cdev *tdev, struct sk_buff *skb, struct
l2t_en
     error = l2t_send(tdev, skb, l2e);
     if (error < 0)
          kfree_skb(skb);
-    return error;
+    return error < 0 ? error : 0;
 }
```

Most languages use exceptions

What are some downsides of exceptions?

Exceptional Exceptions

- Failure modes are hard to spot: *any* function can throw *any* exception at *any* time
- Hard to manage in evolving codebases
- *Especially* hard when manual memory management is involved

Error handling in Rust

- If an *unrecoverable* error occurs, *panic*

```
if sad_times() {
  panic!("Sad times!");
}
```

- If a *recoverable* error may occur, return a `Result`
  - `Result<T, E>` can either be `Ok(`*`some value of type T`*`)` or `Err(`*`some value of type E`*`)`

Usage of Result

```
fn poke_toddler() -> Result<&'static str, &'static str> {
    if get_random_num() > 10 {
        Ok("Hahahaha!")
    } else {
        Err("Waaaaahhh!")
    }
}

fn main() {
    match poke_toddler() {
        Ok(message) => println!("Toddler said: {}", message),
        Err(cry) => println!("Toddler cried: {}", cry),
    }
}
```

unwrap() and expect()

```
// Panic if the baby cries:
let ok_message = poke_toddler().unwrap();
// Same thing, but print a more descriptive panic message:
let ok_message = poke_toddler().expect("Toddler cried :(“);
```

```
// Read line from stdin
let mut line = String::new();
io::stdin().read_line(&mut line).expect("Failed to read from stdin");
```