The Case for Building a Kernel in Rust

Amit Levy[†] Brad Campbell[‡] Branden Ghena[‡]
Pat Pannuto[‡] Philip Levis[†] Prabal Dutta[‡]

Stanford University[†] & University of Michigan[‡]

September 2nd 2017

Memory and type safety bugs plague systems

Name	Description
CVE-2017-9996	The cdxl_decode_frame function in libavcodec/cdxl.c in FFmpeg 2.8.x before 2.8.12, 3.0.x before 3.0.8, 3.1.x before 3.1.8, 3.2.x before 3.2.5, and 3.3.x before 3.3.1 does not exclude the CHUNKY format, which allows remote attackers to cause a denial of service (heap-based buffer overflow and application crash) or possibly have unspecified other impact via a crafted file.
CVE-2017-9995	libavcodec/scpr.c in FFmpeg 3.3 before 3.3.1 does not properly validate height and width data, which allows remote attackers to cause a denial of service (heap-based buffer overflow and application crash) or possibly have unspecified other impact via a crafted file.
CVE-2017-9994	libavcodec/webp.c in FFmpeg before 2.8.12, 3.0.x before 3.0.8, 3.1.x before 3.1.8, 3.2.x before 3.2.5, and 3.3.x before 3.3.1 does not ensure that pix_fmt is set, which allows remote attackers to cause a denial of service (heap-based buffer overflow and application crash) or possibly have unspecified other impact via a crafted file, related to the vp8_decode_mb_row_no_filter and pred8x8_128_dc_8_c functions.
CVE-2017-9992	Heap-based buffer overflow in the decode_dds1 function in libavcodec/dfa.c in FFmpeg before 2.8.12, 3.0.x before 3.0.8, 3.1.x before 3.1.8, 3.2.x before 3.2.5, and 3.3.x before 3.3.1 allows remote attackers to cause a denial of service (application crash) or possibly have unspecified other impact via a crafted file.
CVE-2017-9991	Heap-based buffer overflow in the xwd_decode_frame function in libavcodec/xwddec.c in FFmpeg before 2.8.12, 3.0.x before 3.0.8, 3.1.x before 3.1.8, 3.2.x before 3.2.5, and 3.3.x before 3.3.1 allows remote attackers to cause a denial of service (application crash) or possibly have unspecified other impact via a crafted file.
CVE-2017-9990	Stack-based buffer overflow in the color_string_to_rgba function in libavcodec/xpmdec.c in FFmpeg 3.3 before 3.3.1 allows remote attackers to cause a denial of service (application crash) or possibly have unspecified other impact via a crafted file.
CVE-2017-9987	There is a heap-based buffer overflow in the function hpel_motion in mpegvideo_motion.c in libav 12.1. A crafted input can lead to a remote denial of service attack.

Name	Description
CVE-2017-9762	The cmd_info function in libr/core/cmd_info.c in radare2 1.5.0 allows remote attackers to cause a denial of service (use-after-free and application crash) via a crafted binary file.
CVE-2017-9612	The Ins_IP function in base/ttinterp.c in Artifex Ghostscript GhostXPS 9.21 allows remote attackers to cause a denial of service (use-after-free and application crash) or possibly have unspecified other impact via a crafted document.
CVE-2017-9527	The mark_context_stack function in gc.c in mruby through 1.2.0 allows attackers to cause a denial of service (heap-based use-after-free and application crash) or possibly have unspecified other impact via a crafted .rb file.
CVE-2017-9520	The r_config_set function in libr/config/config.c in radare2 1.5.0 allows remote attackers to cause a denial of service (use-after-free and application crash) via a crafted DEX file.
CVE-2017-9182	libautotrace.a in AutoTrace 0.31.1 allows remote attackers to cause a denial of service (use-after-free and invalid heap read), related to the GET_COLOR function in color.c:16:11.
CVE-2017-8929	The sized_string_cmp function in libyara/sizedstr.c in YARA 3.5.0 allows remote attackers to cause a denial of service (use-after-free and application crash) via a crafted rule.
CVE-2017-8895	In Veritas Backup Exec 2014 before build 14.1.1187.1126, 15 before build 14.2.1180.3160, and 16 before FP1, there is a use-after-free vulnerability in multiple agents that can lead to a denial of service or remote code execution. An authenticated attacker can use this vulnerability to crash the agent or potentially take control of the agent process and then the system it is running on.
CVE-2017-8846	The read_stream function in stream.c in liblrzip.so in lrzip 0.631 allows remote attackers to cause a denial of service (use-after-free and application crash) via a crafted archive.
CVE-2017-8359	Google gRPC before 2017-03-29 has an out-of-bounds write caused by a heap-based use-after-free related to the grpc_call_destroy function in core/lib/surface/call.c.
CVE-2017-8270	In all Qualcomm products with Android releases from CAF using the Linux kernel, a race condition exists in a driver potentially leading to a use-after-free condition.
CVE-2017-8266	In all Qualcomm products with Android releases from CAF using the Linux kernel, a race condition exists in a video driver potentially leading to a use-after-free condition.
CVE-2017-7946	The get_relocs_64 function in libr/bin/format/mach0/mach0.c in radare2 1.3.0 allows remote attackers to cause a denial of service (use-after-free and application crash) via a crafted Mach0 file.

Security: type confusion lead to information leak in decodeURI

Reported by higonggu...@gmail.com, Apr 13 2016

Back to list

VULNERABILITY DETAILS

```
the value passed to function TwoByteSeqStringSetChar maybe not a smi but a HeapObject,
simply casting a point to HeapObject to a smi lead to information leak.
void FullCodeGenerator::EmitTwoByteSeqStringSetChar(CallRuntime* expr) {
  ZoneList<Expression*>* args = expr->arguments();
  DCHECK_EQ(3, args->length());

  Register string = rax;
  Register index = rbx;
  Register value = rcx;

  VisitForStackValue(args->at(0));        // index
  VisitForStackValue(args->at(1));        // value------> maybe point of heap object,
i guess
  VisitForAccumulatorValue(args->at(2));  // string
  PopOperand(value);
  PopOperand(index);
```

Security: type confusion lead to information leak in decodeURI

Reported by higonggu...@gmail.com, Apr 13 2016

Back to list

VULNERABILITY DETAILS

```
the value passed to function TwoByteSeqStringSetChar maybe not a smi but a HeapObject,
simply casting a point to HeapObject to a smi lead to information leak.
void FullCodeGenerator::EmitTwoByteSeqString
  ZoneList<Expression*>* args = expr->argumer
  DCHECK_EQ(3, args->length());

  Register string = rax;
  Register index = rbx;
  Register value = rcx;

  VisitForStackValue(args->at(0));        //
  VisitForStackValue(args->at(1));        //
i guess
  VisitForAccumulatorValue(args->at(2));  //
  PopOperand(value);
  PopOperand(index);
```

MsMpEng: Remotely Exploitable Type Confusion in Windov 10, Windows Server, SCEP, Microsoft Security Essentials, ar

Project Member Reported by taviso@google.com, May 6

Description #5 (taviso@

```
MsMpEng is the Malware Protection service that
is enabled by default on Windows 8, 8.1, 10,
Windows Server 2012, and so on. Additionally, Microsoft Security Es
Endpoint Protection and various other Microsoft security products sl
engine. MsMpEng runs as NT AUTHORITY\SYSTEM without sandboxing, and
without authentication via various Windows services, including Excha

On workstations, attackers can access mpengine by sending emails to
or opening attachments is not necessary), visiting links in a web b
and so on. This level of accessibility is possible because MsMpEng
minifilter to intercept and inspect all system filesystem activity,
contents to anywhere on disk (e.g. caches, temporary internet files
unconfirmed downloads), attachments, etc) is enough to access functi
MIME types and file extensions are not relevant to this vulnerabili
own content identification system.

Vulnerabilities in MsMpEng are among the most severe possible in Wi
privilege, accessibility, and ubiquity of the service.

The core component of MsMpEng responsible for scanning and analysis
Mpengine is a vast and complex attack surface, comprising of handle
archive formats, executable packers and cryptors, full system emula
```

Security: type confusion lead to information leak in decodeURI

Reported by higonggu...@gmail.com, Apr 13 2016

Back to list

VULNERABILITY DETAILS

the value passed to function TwoByteSeqStringSetChar maybe not a smi but a HeapObject, simply casting a point to HeapObject to a smi lead to information leak.

```
void FullCodeGenerator::EmitTwoByteSeqString
  ZoneList<Expression*>* args = expr->argumen
  DCHECK_EQ(3, args->length());

  Register string = rax;
  Register index = rbx;
  Register value = rcx;

  VisitForStackValue(args->at(0));        //
  VisitForStackValue(args->at(1));        //
i guess
  VisitForAccumulatorValue(args->at(2));  //
  PopOperand(value);
  PopOperand(index);
```

MsMpEng: Remotely Exploitable Type Confusion in Window 10, Windows Server, SCEP, Microsoft Security Essentials, ar

Project Member Reported by taviso@google.com, May 6

Description #5 (taviso@

MsMpEng is the Malware Protection service that is enabled by default on Windows 8, 8.1, 10, Windows Server 2012, and so on. Additionally, Microsoft Security Es Endpoint Protection and various other Microsoft security products sh engine. MsMpEng runs as NT AUTHORITY\SYSTEM without sandboxing, and without authentication via various Windows services, including Excha

On workstations, attackers can access mpengine by sending emails to or opening attachments is not necessary), visiting links in a web br and so on. This level of accessibility is possible because MsMpEng l system filesystem activity, hes, temporary internet files tc) is enough to access functi relevant to this vulnerabili

he most severe possible in Wi of the service.

ble for scanning and analysis surface, comprising of handler d cryptors, full system emula

Microsoft Edge and IE: Type confusion in HandleColumnBreakOnColumnSpanningElement

Project Member Reported by ifratric@google.com, Nov 25 2016

PoC:

```
<!-- saved from url=(0014)about:internet -->
<style>
.class1 { float: left; column-count: 5; }
.class2 { column-span: all; columns: 1px; }
table {border-spacing: 0px;}
</style>
<script>
function boom() {
  document.styleSheets[0].media.mediaText = "aaaaaaaaaaaaaaaaaaaa";
  th1.align = "right";
}
</script>
<body onload="setInterval(boom,100)">
<table cellspacing="0">
<tr class="class1">
<th id="th1" colspan="5" width=0></th>
<th class="class2" width=0><div class="class2"></div></th>
```

Note: The analysis below is based on an 64-bit IE (running in single process mode) running on Windows Serv Symbol Server has been down for several days and that's the only configuration for which I had up-to-date Edge and 32-bit IE 11 should behave similarly.

The PoC crashes in

Long history of research

- “Bug finding”
  - Fuzz-ing (1990)
  - DART (2005)
  - KLEE (2008)
  - KINT (2012)
- Type-Safe Kernels:
  - Cedar (1986)
  - Spin (1995)
  - Singularity (2007)

Why are we still building systems in C?

Type safety (typically) isn't free

Type safety usually requires garbage collection.

- Give up control over memory layout and location
- Large trusted runtime
- Either a performance hit or large memory overhead

Type safety (typically) isn't free

Type safety usually requires garbage collection.

- Give up control over memory layout and location
- Large trusted runtime
- Either a performance hit or large memory overhead

Can we use type safety without allowing it to dictate how we design systems?

“Rust is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety.”

-`https://www.rust-lang.org`

1. A 5-minute Introduction to Rust
2. Limitations imposed by Rust
3. Addressing the limitations
4. Case-Study: Tock OS
5. Conclusion & future work

A Systems Builder’s Guide to Rust (abridged)

Rust Features

- Type and memory safe
- Statically enforced type system
- Compiles with the LLVM toolchain to machine code
- C calling convention
- Explicit memory location and layout
- No language runtime

Ownership

Key Property

When the owner goes out of scope, we can deallocate memory for the value.

Ownership

Key Property

When the owner goes out of scope, we can deallocate memory for the value.

Memory for the value `Foo::new()` is allocated and bound to the variable `x`.

```
{
  let x = Foo::new()
}
```

When the scope exits, `x` is no longer valid and the memory is “freed”

Ownership

This is an error:

```
{
  let x = Foo::new();
  let y = x;
  // x not valid here
}
```

because `Foo::new()` has been moved from `x` to `y`, so `x` is no longer valid.

Borrows

```
fn bar(x: &mut Foo) {
  // the borrow is implicitly released.
}

let mut x = Foo::new();
bar(&mut x);
// x still valid here
```

Borrows

```rust
fn bar(x: &mut Foo) {
  // the borrow is implicitly released.
}

let mut x = Foo::new();
bar(&mut x);
// x still valid here
```

Just a pointer at runtime

Borrows

```rust
fn bar(x: &mut Foo) {
  // the borrow is implicitly released.
}

let mut x = Foo::new();
bar(&mut x);
// x still valid here
```

Just a pointer at runtime

- Mutable references (`&mut`) must be unique
- Shared references (`&`) cannot mutate the value

```rust
enum NumOrPointer {
  Num(u32),
  Pointer(&mut u32)
}
```

```rust
enum NumOrPointer {
  Num(u32),
  Pointer(&mut u32)
}
```

```rust
// n.b. will not compile
let external : &mut NumOrPointer;
if let Pointer(internal) = external {
```

```rust
enum NumOrPointer {
  Num(u32),
  Pointer(&mut u32)
}
```

```rust
// n.b. will not compile
let external : &mut NumOrPointer;
if let Pointer(internal) = external {
  *external = Num(0xdeadbeef);
```

```rust
enum NumOrPointer {
  Num(u32),
  Pointer(&mut u32)
}
```

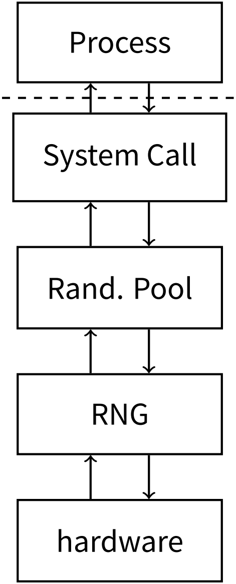
```rust
// n.b. will not compile
let external : &mut NumOrPointer;
if let Pointer(internal) = external {
  *external = Num(0xdeadbeef);
  *internal = 12345;
  // Kaboom: we've just written ‘12345‘
  // to the address ‘0xdeadbeef‘
}
```

```rust
enum NumOrPointer {
  Num(u32),
  Pointer(&mut u32)
}
```

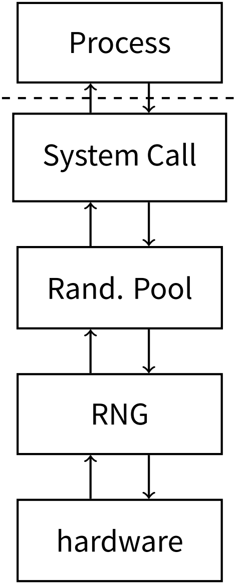
```rust
// n.b. will not compile
let external : &mut NumOrPointer;
if let Pointer(internal) = external {
  *external = Num(0xdeadbeef);
  *internal = 12345;
  // Kaboom: we've just written '12345'
  // to the address '0xdeadbeef'
}
```

```
$ rustc test.rs
error[E0506]: cannot assign to 'external'
  because it is borrowed
```

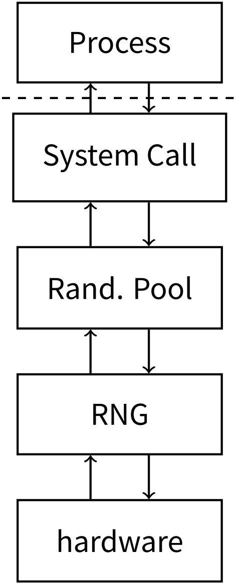
Rust imposed limitations

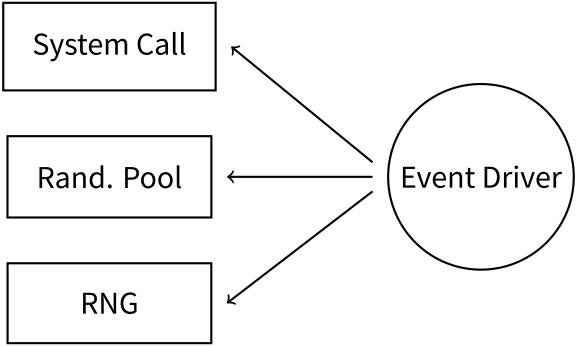

Process
System Call
Rand. Pool
RNG
hardware

```rust
pub struct SysCallDispatcher {
  processes: Vec<Process>,
  pool: &mut RandomPool,
  ...
}

pub struct RandomPool {
  busy: bool,
  pool: Queue<u32>,
  rng: &mut RNG,
  syscall: &mut SysCallDispatcher,
}

pub struct RNG {
  hw_registers: [usize; 16],
  client: &mut RandomPool,
}
```



```
let syscall: SysCallDispatcher;
let pool: RandomPool;
let rng: RNG;

syscalls.pool = &mut poo;
pool.syscall = &mut syscall;
pool.rng = &mut rng;
rng.client = &mut pool;
```




System Call
Rand. Pool
RNG
Event Driver

Interior mutability

It’s actually safe to have mutable aliases in *many cases*.

The key is avoiding mutability and aliasing simultaneously.

Interior mutability

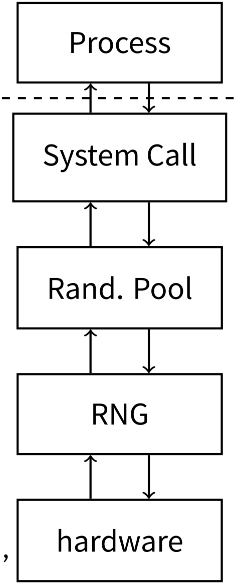
It's actually safe to have mutable aliases in *many cases*.

The key is avoiding mutability and aliasing simultaneously.

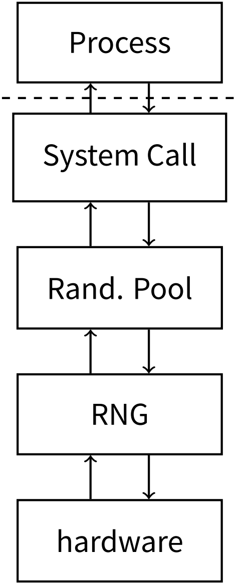
Rust has container types with "interior mutability". Shared references to these types allow mutation, give certain restrictions:

- `Cell`: Only copy-in/out or replace, no references to internal value
- `Mutex`: Gives internal references through mutual-exclusion
- `TakeCell`: Only operates if not already being used

```rust
pub struct SysCallDispatcher {
  processes: TakeCell<Vec<Process>>,
  pool: &RandomPool,
  ...
}

pub struct RandomPool {
  busy: Cell<bool>,
  pool: TakeCell<Queue<u32>>,
  rng: &RNG,
  syscall: &SysCallDispatcher,
}

pub struct RNG {
  hw_registers: TakeCell<[usize; 16]>,
  client: &RandomPool,
}
```



```
let syscall: SysCallDispatcher;
let pool: RandomPool;
let rng: RNG;

syscalls.pool = &poo;
pool.syscall = &syscall;
pool.rng = &rng;
rng.client = &pool;
```



Case study: Tock OS

Tock Overview

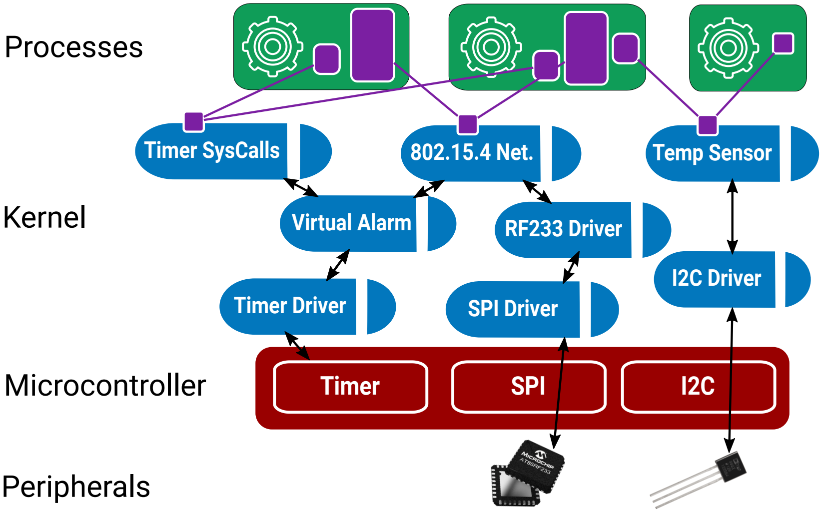
- Security focused embedded operating system

Tock Overview

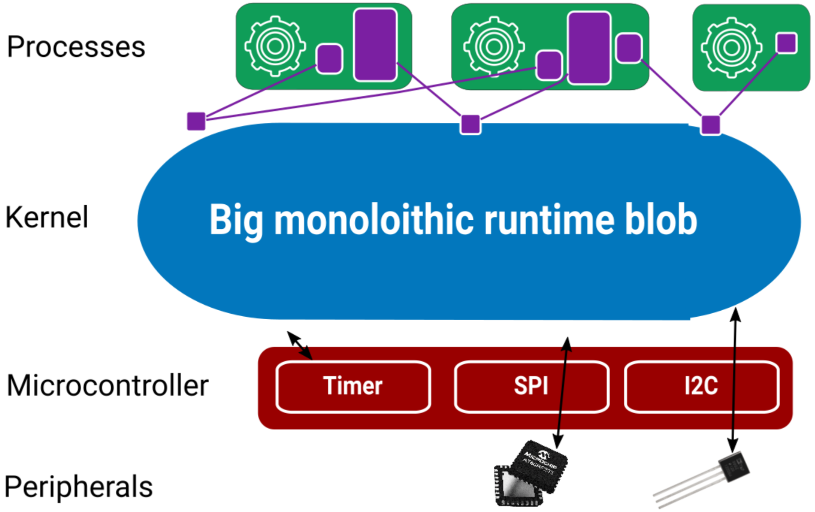
- Security focused embedded operating system
- Kernel components are mostly untrusted

Tock Overview

- Security focused embedded operating system
- Kernel components are mostly untrusted
- Targets microcontrollers with <64kB RAM



- Kernel written in ~26695 lines of Rust



- Kernel written in ~26695 lines of Rust

Example: DMA

```
struct DMAChannel {
  ...
  enabled: Cell<bool>,
  buffer: TakeCell<&'static mut [u8]>,
}
```

Examples: USB

```
enum EpCtl {
  ...
  Enable = 1 << 31,
  ClearNak = 1 << 26,
  Stall = 1 << 21
}
```

```
struct InEndpoint {
  control: Cell<EpCtl>,
  dma_address:
    Cell<&'static DMADescriptor>,
  ...
}
```

```
struct USBRegisters {
  ...
  in_endpoints: Cell<&[InEndpoint; 16]>,
}
```

Minimal TCB

Trusted Kernel components (~3600 LoC)

- Board configuration: 806 LoC
- Process scheduler: 1784 LoC
- Hardware interface: ~1000 LoC

Rust core library

- `Cell`
- `String`, `slice`
- Floating point
- Compiler intrinsics (e.g. `memcpy`)

Conclusion

Type Safety Belongs Everywhere

- Type safety critical tool for building secure systems.

Type Safety Belongs Everywhere

- Type safety critical tool for building secure systems.
- Key question: What is the lowest level of control needed?

Type Safety Belongs Everywhere

- Type safety critical tool for building secure systems.
- Key question: What is the lowest level of control needed?
- *Rust* is an existence proof we can use *today*

Type Safety Belongs Everywhere

- Type safety critical tool for building secure systems.
- Key question: What is the lowest level of control needed?
- *Rust* is an existence proof we can use *today*
- We still need useful performance metrics

Type Safety Belongs Everywhere

- Type safety critical tool for building secure systems.
- Key question: What is the lowest level of control needed?
- *Rust* is an existence proof we can use *today*
- We still need useful performance metrics
- Should experiment with other system types

Type Safety Belongs Everywhere

- Type safety critical tool for building secure systems.
- Key question: What is the lowest level of control needed?
- *Rust* is an existence proof we can use *today*
- We still need useful performance metrics
- Should experiment with other system types
- Can we provide rich security primitives like DIFC?

Type Safety Belongs Everywhere

- Type safety critical tool for building secure systems.
- Key question: What is the lowest level of control needed?
- *Rust* is an existence proof we can use *today*
- We still need useful performance metrics
- Should experiment with other system types
- Can we provide rich security primitives like DIFC?

Type Safety Belongs Everywhere

- Type safety critical tool for building secure systems.
- Key question: What is the lowest level of control needed?
- *Rust* is an existence proof we can use *today*
- We still need useful performance metrics
- Should experiment with other system types
- Can we provide rich security primitives like DIFC?

`amit@amitlevy.com`

`https://tockos.org/`

@talkingtock

```rust
struct App {
    count: u32,
    tx_callback: Callback,
    rx_callback: Callback,
    app_read: Option<AppSlice<Shared, u8>>,
    app_write: Option<AppSlice<Shared, u8>>,
}
pub struct Driver {
    app: TakeCell<App>,
}

driver.app.map(|app| {
    app.count = app.count + 1
});
```

```
/* Load App address into r1, replace with null */
ldr     r1, [r0, 0]
movs    r2, 0
str     r2, [r0, 0]
/* If TakeCell is empty (null) return */
cmp     r1, 0
it      eq
bx      lr
/* Non-null: increment count */
ldr     r2, [r1, 0]
add     r2, r2, 1
str     r2, [r1, 0]
/* Store App back to TakeCell */
str     r1, [r0, 0]
bx      lr
```