Reversing a (Not-so-Simple) Rust Loader

Ringzer0 COUNTERMEASURE Workshop // 2025-11-07

Cindy Xiao (she/her, they/them) // Decoder Loop

Introduction

 Today, we'll be reversing a very straightforward malware loader, written in Rust:

- It has full symbols!
- It does only one thing: load more malware into memory!

However, as we'll see, this loader has a few twists:

- It's written in Rust, and Rust is never easy to reverse.
- It's harder than you might think to find the payload...

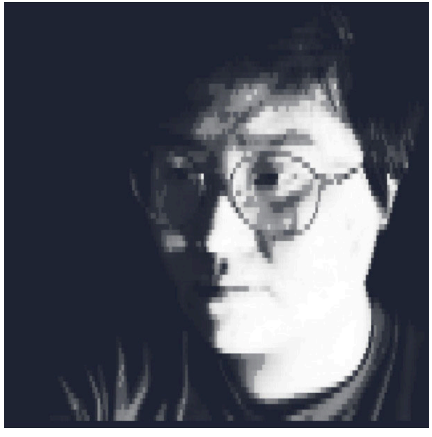
 We'll explore this simple loader together, and use it to learn some key Rust reversing concepts:

- Threads in Rust binaries
- Dynamic dispatch
- Type recovery
- Finding interesting landmarks in Rust binaries

Let's build our confidence in approaching Rust binaries!

My Background

- Formerly a programmer
- Now doing malware analysis
  - Focused on Windows binaries
  - Symbols? What symbols?
- Founder of *Decoder Loop* (decoderloop.com)
  - Training you how to reverse binaries written in modern programming languages (like Rust!)
  - Interested in bringing Rust RE training to your conference? Come talk to me!



Your Background?

✋ How many of you here:

1. Have ever reversed a Rust binary?
2. Have ever reversed malware?
3. Have ever used Binary Ninja?

Asking questions, getting resources

If you have trouble with:

- Seeing the materials (font size, colours, etc.)
- Hearing me speak

✋ Feel free to interrupt and let me know right away!

If you miss something during the workshop:

- Workshop repository: github.com/decoderloop/2025-11-07-ringzer0-countermeasure-not-so-simple-rust-loader-workshop
  - Slides
  - Sample Binary Ninja Database
- Original article for following along: cxiao.net - Reversing a (not-so-) Simple Rust Loader

What you'll need

- Binary Ninja Free: binary.ninja/free
- Malware sample: Malware Bazaar download
- Static analysis only today
  - However, take precautions to not accidentally execute

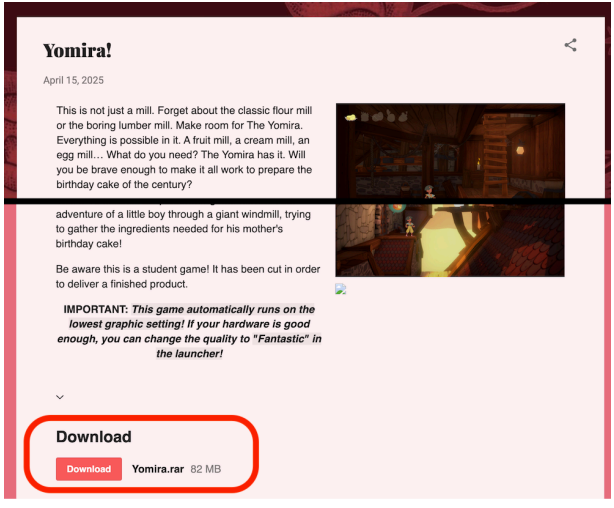
Optional materials:

- Workshop repository: github.com/decoderloop/2025-11-07-ringzer0-countermeasure-not-so-simple-rust-loader-workshop
  - Slides
  - Sample Binary Ninja Database
- Original article for following along: cxiao.net - Reversing a (not-so-) Simple Rust Loader

Where does this sample come from?

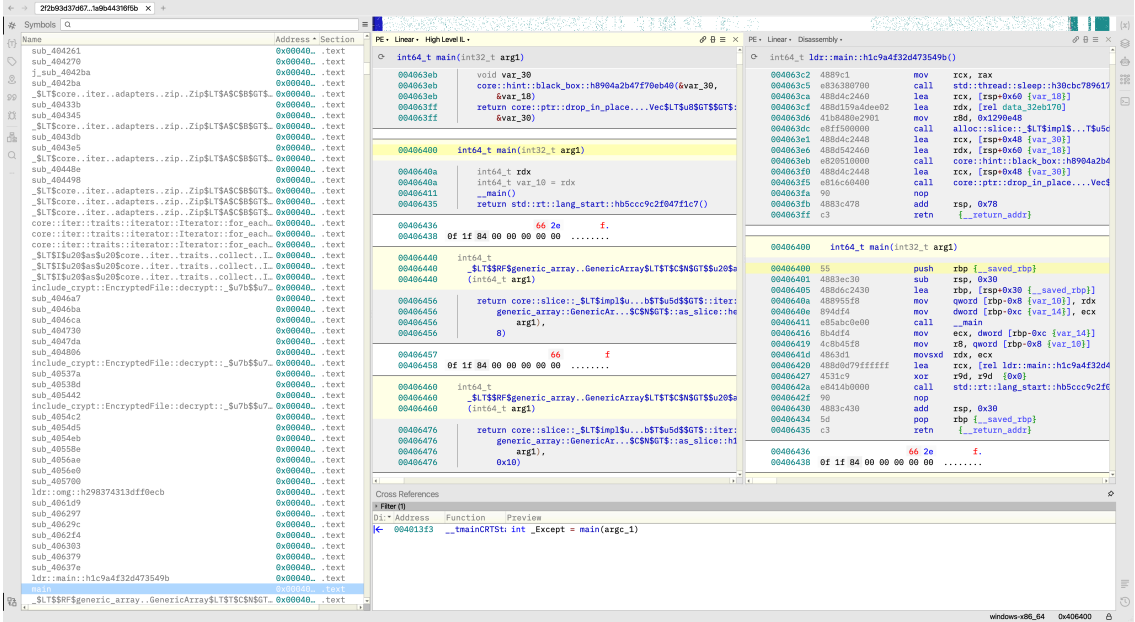
Malware delivered via downloads from websites linked from Discord messages. Starts with a victim receiving a message on Discord from another user:

> Can you try this game I just made?



Triage

First look in Binary Ninja



Strings

Panic messages from the Rust standard library.

```
Wfrom_str_radix_int: must lie in the range `[2, 36]` - found
encode_utf8: need  bytes to encode U+ but buffer has just
encode_utf16: need
attempt to divide by zero
attempt to calculate the remainder with a divisor of zero
coroutine resumed after completion
`async fn` resumed after completion
`async gen fn` resumed after completion
```

Source file paths from the Rust standard library.

```
/rustc/4eb161250e340c8f48f66e2b929ef4a5bed7c181\library\core\src\num\mod.rs
/rustc/4eb161250e340c8f48f66e2b929ef4a5bed7c181\library\core\src\ub_checks.rs
/rustc/4eb161250e340c8f48f66e2b929ef4a5bed7c181\library\core\src\str\mod.rs
/rustc/4eb161250e340c8f48f66e2b929ef4a5bed7c181\library\std\src\io\mod.rs
```

Strings

Source file paths from third-party Rust libraries.

```
/root/.cargo/registry/src/index.crates.io-1949cf8c6b5b557f/include-crypt-crypto-0.1.0/src/xor.rs
/root/.cargo/registry/src/index.crates.io-1949cf8c6b5b557f/obfstr-0.2.6/src/bytes.rs
/root/.cargo/registry/src/index.crates.io-1949cf8c6b5b557f/aes-soft-0.6.4/src/impls.rs
/root/.cargo/registry/src/index.crates.io-1949cf8c6b5b557f/hex-0.4.3/src/lib.rs
```

Strings

The malware author's own source files.

```
mexec/src/lib.rs
mexec/src/peloader/mod.rs
mexec/src/peparser/header.rs
mexec/src/peloader/winapi.rs
mexec/src/peparser/pe.rs
mexec/src/peparser/section.rs
```

Strings

Symbol information in the binary.

> **Tip**
>
> Malware authors often still forget to strip debug symbols from Rust binaries, because the `release` build profile of the Rust toolchain still includes debug symbols by default!

```
_ZN4core6option15Option$LT$T$GT$6as_mut17h61005c7a76b5b550E
_ZN87_$LT$core..str..iter..EncodeUtf16$u20$as$u20$core..iter..trai
_ZN4core3ops8function6FnOnce9call_once17h644472653fc53b71E
_ZN78_$LT$core..ptr..non_null..NonNull$LT$T$GT$$u20$as$u20$core..c
_ZN91_$LT$core..slice..iter..Iter$LT$T$GT$$u20$as$u20$core..iter..
```

A note on demangling symbols

Tip

Binary Ninja does not correctly demangle Rust symbols right now!

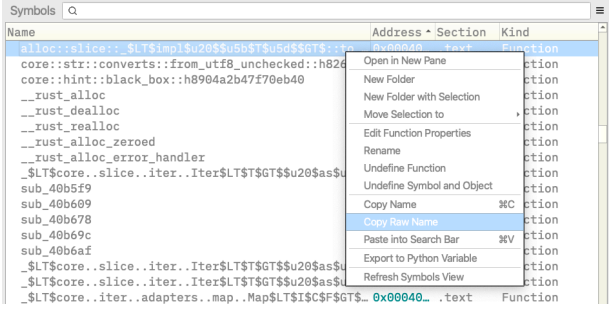
If you're using Binary Ninja Free:

1. Symbols pane > Right click on the symbol of interest > *Copy Raw Name*
2. Paste into the Rust Demangler tool at https://saidinesh5.github.io/rust-demangler-online/
3. Copy demangled name and use to rename function

Tip

If you're using a paid version of Binary Ninja:

Use inspier's Rust Demangle plugin, available in the Plugin Manager.



Finding the entry point

```
int64_t main(int32_t arg1)

00406400    int64_t main(int32_t arg1)

00406400  55                push    rbp {__saved_rbp}
00406401  4883ec30          sub     rsp, 0x30
00406405  488d6c2430        lea     rbp, [rsp+0x30 {__saved_rbp}]
0040640a  488955f8          mov     qword [rbp-0x8 {var_10}], rdx
0040640e  894df4            mov     dword [rbp-0xc {var_14}], ecx
00406411  e85abc0e00        call    __main
00406416  8b4df4            mov     ecx, dword [rbp-0xc {var_14}]
00406419  4c8b45f8          mov     r8, qword [rbp-0x8 {var_10}]
0040641d  4863d1            movsxd  rdx, ecx
00406420  488d0d79ffffff    lea     rcx, [rel ldr::main::h1c9a4f32d473549b]
00406427  4531c9            xor     r9d, r9d  {0x0}
0040642a  e8414b0000        call    std::rt::lang_start::hb5ccc9c2f047f1c7
0040642f  90                nop
00406430  4883c430          add     rsp, 0x30
00406434  5d                pop     rbp {__saved_rbp}
00406435  c3                retn     {__return_addr}
```

Finding the entry point

Rust runtime source code: `std::rt::lang_start` function

```rust
#[lang = "start"]
fn lang_start<T: crate::process::Termination + 'static>(
    main: fn() -> T,
    argc: isize,
    argv: *const *const u8,
    sigpipe: u8,
) -> isize {
    let Ok(v) = lang_start_internal(
        &move || crate::sys::backtrace::__rust_begin_short_backtrace(main).report().to_i32(),
        argc,
        argv,
        sigpipe,
    );
    v
}
```

Diagram on `std::rt::lang_start`, and its relation to the C runtime & the programmer's code

Reversing `ldr::main`

Reversing `ldr::main`

In the decompilation of `ldr::main`, a couple of features stand out:

- There is a call to the Rust standard library thread creation function `std::thread::spawn`.
- However, it's not clear where the entry point to the new thread is.

Reversing `ldr::main`

There is a call to

`alloc::slice::<impl [T]>::to_vec`

This copies a slice of data into a new `std::vec::Vec`.

The `vec::Vec` type

`std::vec::Vec` is a growable array, with each array element containing the same data type.

- `len` field: The actual amount of elements currently inside the `Vec`.
- `capacity` field: The total amount of elements that *could* fit inside the space available in the `Vec`'s backing memory.
- Similar to C++ standard library's `std::vector`!

std::vec::Vec<u8>		
00	buf: RawVec<u8>	ptr: *u8
08		cap: usize
10	len: usize	

Tip

For Rust reverse engineers looking up Rust standard library information:

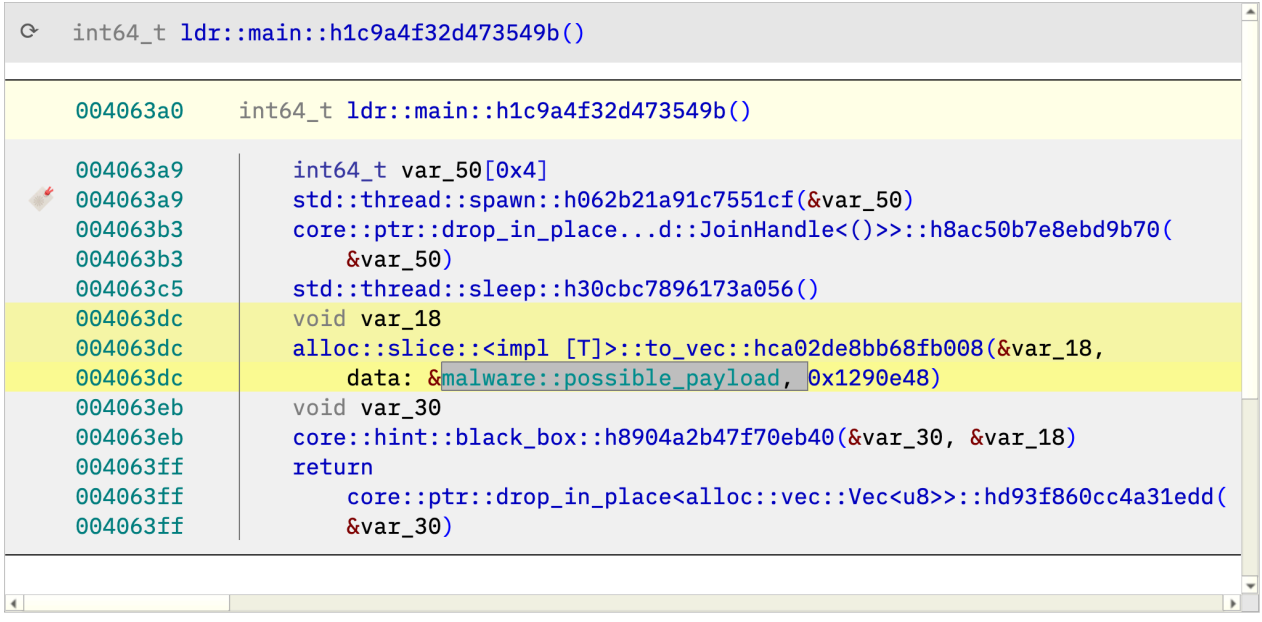
Use stdrs.dev, rather than the regular Rust standard library documentation at doc.rust-lang.org/std.

stdrs.dev is a version of the Rust standard library docs that includes internal implementation details and non-public functions.

A payload at `0x32eb170` ?

```
0x32eb160 .rdata {0x4f5000-0x45a5d20} Read-only data

032eb160  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
032eb170  data_32eb170:
032eb170  4d 5a 90 00 03 00 00 00-04 00 00 00 ff ff 00 00  MZ..............
032eb180  b8 00 00 00 00 00 00 00-40 00 00 00 00 00 00 00  ........@.......
032eb190  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
032eb1a0  00 00 00 00 00 00 00 00-00 00 00 00 38 01 00 00  ............8...
032eb1b0  0e 1f ba 0e 00 b4 09 cd-21 b8 01 4c cd 21 54 68  ........!..L.!Th
032eb1c0  69 73 20 70 72 6f 67 72-61 6d 20 63 61 6e 6e 6f  is program canno
032eb1d0  74 20 62 65 20 72 75 6e-20 69 6e 20 44 4f 53 20  t be run in DOS
032eb1e0  6d 6f 64 65 2e 0d 0d 0a-24 00 00 00 00 00 00 00  mode....$.......
032eb1f0  14 f9 01 f5 50 98 6f a6-50 98 6f a6 50 98 6f a6  ....P.o.P.o.P.o.
032eb200  1b e0 6c a7 59 98 6f a6-1b e0 6a a7 9a 98 6f a6  ..l.Y.o...j...o.
032eb210  1b e0 6b a7 42 98 6f a6-41 1e 92 a6 54 98 6f a6  ..k.B.o.A...T.o.
032eb220  41 1e 6c a7 5a 98 6f a6-41 1e 6b a7 42 98 6f a6  A.l.Z.o.A.k.B.o.
032eb230  41 1e 6a a7 36 98 6f a6-59 e0 fc a6 52 98 6f a6  A.j.6.o.Y...R.o.
032eb240  cf 90 14 a6 52 98 6f a6-1b e0 6e a7 5b 98 6f a6  ....R.o...n.[.o.
032eb250  50 98 6e a6 bf 98 6f a6-d6 1e 6a a7 54 98 6f a6  P.n...o...j.T.o.
032eb260  d6 1e 6b a7 70 98 6f a6-d6 1e 6f a7 51 98 6f a6  ..k.p.o...o.Q.o.
032eb270  d6 1e 90 a6 51 98 6f a6-50 98 f8 a6 51 98 6f a6  ....Q.o.P...Q.o.
032eb280  d6 1e 6d a7 51 98 6f a6-52 69 63 68 50 98 6f a6  ..m.Q.o.RichP.o.
032eb290  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
032eb2a0  00 00 00 00 00 00 00 00-50 45 00 00 64 86 0d 00  ........PE..d...
```

Creating a `Vec`



Creating a `Vec`

`alloc::slice::<impl [T]>::to_vec`

```
int64_t* alloc::slice::<impl [T]>::to_vec::hca02de8bb68fb008(int64_t* arg1, uint8_t* d

0040b4e0    int64_t* alloc::slice::<impl [T]>::to_vec::hca02de8bb68fb008(
0040b4e0      int64_t* arg1, uint8_t* data, int64_t arg3)

0040b4ec        <T as alloc::slice::hack::ConvertVec>::to_vec::hb9344dbcba8d97d4(
0040b4ec            arg1, data, arg3)
0040b4fa        return arg1
```

Creating a `Vec`

`<T as alloc::slice::hack::ConvertVec>::to_vec`

```
int64_t* <T as alloc::slice::hack::ConvertVec>::to_vec::hb9344dbcba8d97d4(int64_t* arg1,

004108b0    int64_t* <T as alloc::slice::hack::ConvertVec>::to_vec::hb9344dbcba8d97d4(
004108b0      int64_t* arg1, uint8_t* data, int64_t arg3)

004108e0      int64_t rax_1
004108e0      int64_t rdx
004108e0      rax_1, rdx =
004108e0          alloc::raw_vec::RawVecIn...::with_capacity_in::h5f996dd967e84c92(
004108e0          arg3, 1, 1)
004108ef      int64_t var_8 = 0
00410915      memcpy(dest: rdx, src: data, n: arg3)
00410933      *arg1 = rax_1
0041093b      arg1[1] = rdx
00410944      arg1[2] = arg3
0041094c      return arg1
```

Defining a `Vec` type

```
int64_t* <T as alloc::slice::hack::ConvertVec>::to_vec::hb9344dbcba8d97d4(int64_t* arg1,

004108b0    int64_t* <T as alloc::slice::hack::ConvertVec>::to_vec::hb9344dbcba8d97d4(
004108b0      int64_t* arg1, uint8_t* data, int64_t arg3)

004108e0        int64_t rax_1
004108e0        int64_t rdx
004108e0        rax_1, rdx =
004108e0            alloc::raw_vec::RawVecIn...::with_capacity_in::h5f996dd967e84c92(
004108e0            arg3, 1, 1)
004108ef        int64_t var_8 = 0
00410915        memcpy(dest: rdx, src: data, n: arg3)
00410933        *arg1 = rax_1
0041093b        arg1[1] = rdx
00410944        arg1[2] = arg3
0041094c        return arg1
```

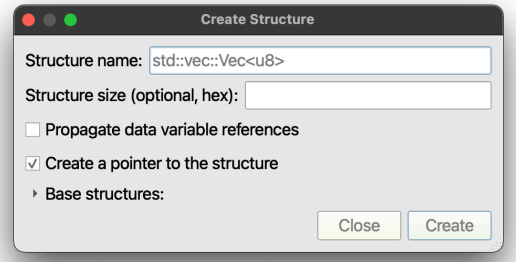
std::vec::Vec<u8>		
00	buf: RawVec<u8>	ptr: *u8
08		cap: usize
10	len: usize	

Defining a `Vec` type

Highlight `arg1` > Press `S`, to create a new Structure type

- Name the structure `std::vec::Vec<u8>`
- Check "Create a pointer to the structure"

This creates the new structure, and sets the type of `arg1` as a pointer to the newly created structure.



Tip

In Binary Ninja, you can have types and names that contain special characters (spaces, angle brackets, `::`, etc.).

In some windows, you may need to surround the name with backticks (the ` character)

Defining a `Vec` type

Highlight `arg1` again > Press `S` to automatically create fields from accessed memory offsets

```
struct std::Vec::vec<u8>* <T as alloc::slice::hack::ConvertVec>::to_vec::hb9344dbcba8d97d4(struc

004108b0    struct std::Vec::vec<u8>*
004108b0      <T as alloc::slice::hack::ConvertVec>::to_vec::hb9344dbcba8d97d4(
004108b0      struct std::Vec::vec<u8>* arg1, uint8_t* data, int64_t arg3)

004108e0      int64_t rax_1
004108e0      int64_t rdx
004108e0      rax_1, rdx = alloc::raw_vec::RawVecIn...::with_capacity_in::h5f996dd967e84c92(
004108e0          arg3, 1, 1)
004108ef      int64_t var_8 = 0
00410915      memcpy(dest: rdx, src: data, n: arg3)
00410933      arg1->__offset(0x0).q = rax_1
0041093b      arg1->__offset(0x8).q = rdx
00410944      arg1->__offset(0x10).q = arg3
0041094c      return arg1
```

std::vec::Vec<u8>		
00	buf: RawVec<u8>	ptr: *u8
08		cap: usize
10	len: usize	

Defining a `Vec` type

Highlight `field_8` > Press `N` to rename it to `ptr`

```
struct std::vec::Vec* <T as alloc::slice::hack::ConvertVec>::to_vec::hb9344dbcba8d97d4(struct st

004108b0    struct std::vec::Vec*
004108b0      <T as alloc::slice::hack::ConvertVec>::to_vec::hb9344dbcba8d97d4(
004108b0      struct std::vec::Vec* arg1, uint8_t* data, int64_t arg3)

004108e0      int64_t rax_1
004108e0      int64_t data_ptr
004108e0      rax_1, data_ptr =
004108e0          alloc::raw_vec::RawVecIn...::with_capacity_in::h5f996dd967e84c92(arg3, 1,
004108e0          1)
004108ef      int64_t var_8 = 0
00410915      memcpy(dest: data_ptr, src: data, n: a
00410933      arg1->field_0 = rax_1
0041093b      arg1->ptr = data_ptr
00410944      arg1->field_10 = arg3
0041094c      return arg1
```

std::vec::Vec<u8>		
00	buf: RawVec<u8>	ptr: *u8
08		cap: usize
10	len: usize	

Defining a `Vec` type

```
struct std::vec::Vec* <T as alloc::slice::hack::ConvertVec>::to_vec::hb9344dbcba8d97d4(struct st

004108b0    struct std::vec::Vec*
004108b0      <T as alloc::slice::hack::ConvertVec>::to_vec::hb9344dbcba8d97d4(
004108b0      struct std::vec::Vec* arg1, uint8_t* data, int64_t data_len)

004108e0      int64_t rax_1
004108e0      int64_t data_ptr
004108e0      rax_1, data_ptr =
004108e0          alloc::raw_vec::RawVecIn...::with_capacity_in::h5f996dd967e84c92(data_len,
004108e0          1, 1)
004108ef      int64_t var_8 = 0
00410915      memcpy(dest: data_ptr, src: data, n: data_len)
00410933      arg1->field_0 = rax_1
0041093b      arg1->ptr = data_ptr
00410944      arg1->len = data_len
0041094c      return arg1
```

std::vec::Vec<u8>		
00	buf: RawVec<u8>	ptr: *u8
08		cap: usize
10	len: usize	

Multiple return values

```
struct std::vec::Vec* <T as alloc::slice::hack::ConvertVec>::to_vec::hb9344dbcba8d97d4(stru

004108b0    struct std::vec::Vec*
004108b0      <T as alloc::slice::hack::ConvertVec>::to_vec::hb9344dbcba8d97d4(
004108b0      struct std::vec::Vec* arg1, uint8_t* data, int64_t data_len)

004108e0        int64_t rax_1
004108e0        int64_t data_ptr
004108e0        rax_1, data_ptr =
004108e0            alloc::raw_vec::RawVecIn...::with_capacity_in::h5f996dd967e84c92(
004108e0            data_len, 1, 1)
004108ef        int64_t var_8 = 0
00410915        memcpy(dest: data_ptr, src: data, n: data_len)
00410933        arg1->field_0 = rax_1
0041093b        arg1->ptr = data_ptr
00410944        arg1->len = data_len
0041094c        return arg1
```

Multiple return values

```
struct std::vec::Vec* <T as alloc::slice::hack::ConvertVec>::to_vec::hb9344dbcba8d97d4(struct std::vec::
```

```
004108b0    struct std::vec::Vec* <T as alloc::slice::hack::ConvertVec>::to_vec::hb9344dbcba8d97d4(
004108b0      struct std::vec::Vec* arg1, uint8_t* data, int64_t data_len)

004108b0  4883ec68            sub     rsp, 0x68
004108b4  4c89442430          mov     qword [rsp+0x30 {var_38}], r8
004108b9  4889542428          mov     qword [rsp+0x28 {var_40}], rdx
004108be  4889c8              mov     rax, rcx
004108c1  488b4c2430          mov     rcx, qword [rsp+0x30 {var_38}]
004108c6  4889442438          mov     qword [rsp+0x38 {var_30}], rax
004108cb  4889442440          mov     qword [rsp+0x40 {var_28}], rax
004108d0  41b801000000        mov     r8d, 0x1
004108d6  4c8d0d1bc91604      lea     r9, [rel data_457d1f8]
004108dd  4c89c2              mov     rdx, r8  {0x1}
004108e0  e86b000000          call    alloc::raw_vec::RawVecIn...::with_capacity_in::h5f996dd967e84c92
004108e5  4889442450          mov     qword [rsp+0x50 {var_18}], rax
004108ea  4889542458          mov     qword [rsp+0x58 {var_10}], rdx
004108ef  48c7442460000000…   mov     qword [rsp+0x60 {var_8}], 0x0
004108f8  488b442458          mov     rax, qword [rsp+0x58 {var_10}]
004108fd  4889442448          mov     qword [rsp+0x48 {var_20}], rax
```

Multiple return values

```
int64_t alloc::raw_vec::RawVecInner<A>::with_capacity_in::h5f996dd967e84c92(int64_t arg1, int64_
004109b9  4c8b442430          mov     r8, qword [rsp+0x30 {var_58}]
004109be  488b4c2450          mov     rcx, qword [rsp+0x50 {result}]
004109c3  488b542458          mov     rdx, qword [rsp+0x58 {var_30}]
004109c8  e8a8d70b00          call    alloc::raw_vec::handle_error::h539d02c6e8ead8b6
{ Does not return }

004109cd  48c7842480000000…   mov     qword [rsp+0x80 {var_8}], 0xffffffffffffffff
004109d9  eb0d                jmp     0x4109e8

004109db  488b442460          mov     rax, qword [rsp+0x60 {var_28}]
004109e0  4889842480000000    mov     qword [rsp+0x80 {result_1}], rax

004109e8  488b442460          mov     rax, qword [rsp+0x60 {var_28}]
004109ed  488b542468          mov     rdx, qword [rsp+0x68 {var_20}]
004109f2  4881c488000000      add     rsp, 0x88
004109f9  c3                  retn     {__return_addr}
```

In theory:

- For Rust code calling other Rust code: The calling convention is neither *documented* nor *stable*.
  - Rust programmers can specify a calling convention for a given function to use, but this is for interfacing with non-Rust code (FFI)
- It's not guaranteed that function calls will follow the platform calling convention.

In practice:

- For arguments: Usually, the compiler does actually place arguments in the registers / locations specified in the platform calling convention.
  - Windows x64 ABI: `rcx`, `rdx`, `r8`, etc.
- A single argument in the Rust source code may be split across multiple register locations, though!
- As we see here, multiple return values are also possible.

> **Tip**
> Ghidra has a `__rustcall` for Rust Unix binaries. Do not trust this; there is no such calling convention as `__rustcall`!

Defining a `Vec` type: Capacity

```
struct std::vec::Vec* <T as alloc::slice::hack::ConvertVec>::to_vec::hb9344dbcba8d97d4(struct st

004108b0    struct std::vec::Vec*
004108b0      <T as alloc::slice::hack::ConvertVec>::to_vec::hb9344dbcba8d97d4(
004108b0      struct std::vec::Vec* arg1, uint8_t* data, int64_t data_len)

004108e0        int64_t allocated_vec_capacity
004108e0        int64_t data_ptr
004108e0        allocated_vec_capacity, data_ptr =
004108e0            alloc::raw_vec::RawVecIn...::with_capacity_in::h5f996dd967e84c92(data_len,
004108e0            1, 1)
004108ef        int64_t var_8 = 0
00410915        memcpy(dest: data_ptr, src: data, n: data_len)
00410933        arg1->cap = allocated_vec_capacity
0041093b        arg1->ptr = data_ptr
00410944        arg1->len = data_len
0041094c        return arg1
```

std::vec::Vec<u8>		
00	buf: RawVec<u8>	ptr: *u8
08		cap: usize
10	len: usize	

Our allocated `Vec`

```
int64_t ldr::main::h1c9a4f32d473549b()

004063a0    int64_t ldr::main::h1c9a4f32d473549b()

004063a9        int64_t var_50[0x4]
004063a9        std::thread::spawn::h062b21a91c7551cf(&var_50)
004063b3        core::ptr::drop_in_place...d::JoinHandle<()>>::h8ac50b7e8ebd9b70(&var_50)
004063c5        std::thread::sleep::h30cbc7896173a056()
004063dc        struct std::vec::Vec allocated_vec
004063dc        alloc::slice::<impl [T]>::to_vec::hca02de8bb68fb008(&allocated_vec,
004063dc            data: &data_32eb170, data_len: 0x1290e48)
004063eb        void var_30
004063eb        core::hint::black_box::h8904a2b47f70eb40(&var_30, &allocated_vec)
004063ff        return core::ptr::drop_in_place<alloc::vec::Vec<u8>>::hd93f860cc4a31edd(
004063ff            &var_30)
```

A payload at `0x32eb170`, with length `0x1290e48`

```
0x32eb160 .rdata {0x4f5000-0x45a5d20} Read-only data

032eb160  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
032eb170  data_32eb170:
032eb170  4d 5a 90 00 03 00 00 00-04 00 00 00 ff ff 00 00  MZ..............
032eb180  b8 00 00 00 00 00 00 00-40 00 00 00 00 00 00 00  ........@.......
032eb190  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
032eb1a0  00 00 00 00 00 00 00 00-00 00 00 00 38 01 00 00  ............8...
032eb1b0  0e 1f ba 0e 00 b4 09 cd-21 b8 01 4c cd 21 54 68  ........!..L.!Th
032eb1c0  69 73 20 70 72 6f 67 72-61 6d 20 63 61 6e 6e 6f  is program canno
032eb1d0  74 20 62 65 20 72 75 6e-20 69 6e 20 44 4f 53 20  t be run in DOS
032eb1e0  6d 6f 64 65 2e 0d 0d 0a-24 00 00 00 00 00 00 00  mode....$.......
032eb1f0  14 f9 01 f5 50 98 6f a6-50 98 6f a6 50 98 6f a6  ....P.o.P.o.P.o.
032eb200  1b e0 6c a7 59 98 6f a6-1b e0 6a a7 9a 98 6f a6  ..l.Y.o...j...o.
032eb210  1b e0 6b a7 42 98 6f a6-41 1e 92 a6 54 98 6f a6  ..k.B.o.A...T.o.
032eb220  41 1e 6c a7 5a 98 6f a6-41 1e 6b a7 42 98 6f a6  A.l.Z.o.A.k.B.o.
032eb230  41 1e 6a a7 36 98 6f a6-59 e0 fc a6 52 98 6f a6  A.j.6.o.Y...R.o.
032eb240  cf 90 14 a6 52 98 6f a6-1b e0 6e a7 5b 98 6f a6  ....R.o...n.[.o.
032eb250  50 98 6e a6 bf 98 6f a6-d6 1e 6a a7 54 98 6f a6  P.n...o...j.T.o.
032eb260  d6 1e 6b a7 70 98 6f a6-d6 1e 6f a7 51 98 6f a6  ..k.p.o...o.Q.o.
032eb270  d6 1e 90 a6 51 98 6f a6-50 98 f8 a6 51 98 6f a6  ....Q.o.P...Q.o.
032eb280  d6 1e 6d a7 51 98 6f a6-52 69 63 68 50 98 6f a6  ..m.Q.o.RichP.o.
032eb290  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
032eb2a0  00 00 00 00 00 00 00 00-50 45 00 00 64 86 0d 00  ........PE..d...
```

Carving out the payload at `0x32eb170`, with length `0x1290e48`

```
cc6e574bbed3faba611ea5521a5db4ff4a87500dde7f180fc81c88e7d834d9f6
```

```
Header.Machine           : AMD64
Header.Subsystem         : Windows GUI
Header.MinimumOS         : Windows Vista
Header.ExportName        : RAC.exe
Header.Type              : EXE
Header.Bits              : 64
Header.ImageBase         : 0x0000000140000000
Header.ImageSize         : 29278072
Version.CompanyName      : RIDEV YAZILIM SİSTEMLERİ LTD ŞTİ
Version.FileDescription  : CraftRise
Version.FileVersion      : 2.0.0.0
Version.InternalName     : CraftRise
Version.LegalCopyright   : Copyright © RIDEV YAZILIM SİSTEMLERİ LTD ŞTİ 2023
Version.OriginalFilename : CraftRise
Version.ProductName      : CraftRise
Version.ProductVersion   : 2.0.0.0
Version.LangID           : 040904B0
Version.Charset          : Unicode
Version.Language         : English (United States)
TimeStamp.Linker         : 2025-03-18 10:57:30
TimeStamp.Export         : 2106-02-07 06:28:15
```

Where does the allocated Vec get used?

```
int64_t ldr::main::h1c9a4f32d473549b()

004063a0    int64_t ldr::main::h1c9a4f32d473549b()

004063a9        int64_t var_50[0x4]
004063a9        std::thread::spawn::h062b21a91c7551cf(&var_50)
004063b3        core::ptr::drop_in_place...d::JoinHandle<()>>::h8ac50b7e8ebd9b70(&var_50)
004063c5        std::thread::sleep::h30cbc7896173a056()
004063dc        struct std::vec::Vec allocated_vec
004063dc        alloc::slice::<impl [T]>::to_vec::hca02de8bb68fb008(&allocated_vec,
004063dc            data: &data_32eb170, data_len: 0x1290e48)
004063eb        void var_30
004063eb        core::hint::black_box::h8904a2b47f70eb40(&var_30, &allocated_vec)
004063ff        return core::ptr::drop_in_place<alloc::vec::Vec<u8>>::hd93f860cc4a31edd(
004063ff            &var_30)
```

Looking inside `core::hint::black_box`

```
int64_t* core::hint::black_box::h8904a2b47f70eb40(int64_t* arg1, int64_t* arg2)

0040b510    int64_t* core::hint::black_box::h8904a2b47f70eb40(int64_t* arg1, int64_t* arg2)

0040b516        *arg1 = *arg2
0040b51d        arg1[1] = arg2[1]
0040b525        arg1[2] = arg2[2]
0040b529        return arg1
```

What is `core::hint::black_box`?

`core::hint::black_box`

> An identity function that ***hints*** to the compiler to be maximally pessimistic about what `black_box` could do.
> ...
> In practice, `black_box` serves two purposes:
> 1. It prevents the compiler from making optimizations related to the value returned by `black_box`
> 2. It forces the value passed to `black_box` to be calculated, even if the return value of `black_box` is unused

Our payload is a decoy!

- The allocated `Vec` gets passed into `core::hint::black_box` and never used!
- `core::hint::black_box` is here to prevent the useless `Vec` from simply being optimized away

```
int64_t ldr::main::h1c9a4f32d473549b()

004063a0    int64_t ldr::main::h1c9a4f32d473549b()

004063a9        int64_t var_50[0x4]
004063a9        std::thread::spawn::h062b21a91c7551cf(&var_50)
004063b3        core::ptr::drop_in_place...d::JoinHandle<()>>::h8ac50b7e8ebd9b70(&var_50)
004063c5        std::thread::sleep::h30cbc7896173a056()
004063dc        struct std::vec::Vec allocated_vec
004063dc        alloc::slice::<impl [T]>::to_vec::hca02de8bb68fb008(&allocated_vec,
004063dc            data: &data_32eb170, data_len: 0x1290e48)
004063eb        void var_30
004063eb        core::hint::black_box::h8904a2b47f70eb40(&var_30, &allocated_vec)
004063ff        return core::ptr::drop_in_place<alloc::vec::Vec<u8>>::hd93f860cc4a31edd(
004063ff            &var_30)
```

Finding the actual payload

A call to `std::thread::spawn`

```
int64_t ldr::main::h1c9a4f32d473549b()

004063a0    int64_t ldr::main::h1c9a4f32d473549b()

004063a9        int64_t var_50[0x4]
004063a9        std::thread::spawn::h062b21a91c7551cf(&var_50)
004063b3        core::ptr::drop_in_place...d::JoinHandle<()>>::h8ac50b7e8ebd9b70(&var_50)
004063c5        std::thread::sleep::h30cbc7896173a056()
004063dc        struct std::vec::Vec allocated_vec
004063dc        alloc::slice::<impl [T]>::to_vec::hca02de8bb68fb008(&allocated_vec,
004063dc            data: &data_32eb170, data_len: 0x1290e48)
004063eb        void var_30
004063eb        core::hint::black_box::h8904a2b47f70eb40(&var_30, &allocated_vec)
004063ff        return core::ptr::drop_in_place<alloc::vec::Vec<u8>>::hd93f860cc4a31edd(
004063ff            &var_30)
```

`std::thread::spawn`

Rust standard library docs: `std::thread::spawn`

```rust
pub fn spawn<F, T>(f: F) -> JoinHandle<T>
where
    F: FnOnce() -> T + Send + 'static,
    T: Send + 'static,
```

```rust
use std::thread;

let computation = thread::spawn(|| {
    // Some expensive computation.
    42
});

let result = computation.join().unwrap();
println!("{result}");
```

Where's the thread entry point?

Somewhere, there should be a pointer to some code that's executed when the thread spawns.

```
int64_t ldr::main::h1c9a4f32d473549b()

004063a0    int64_t ldr::main::h1c9a4f32d473549b()

004063a9        int64_t var_50[0x4]
004063a9        std::thread::spawn::h062b21a91c7551cf(&var_50)
004063b3        core::ptr::drop_in_place...d::JoinHandle<()>>::h8ac50b7e8ebd9b70(&var_50)
004063c5        std::thread::sleep::h30cbc7896173a056()
004063dc        struct std::vec::Vec allocated_vec
004063dc        alloc::slice::<impl [T]>::to_vec::hca02de8bb68fb008(&allocated_vec,
004063dc            data: &data_32eb170, data_len: 0x1290e48)
004063eb        void var_30
004063eb        core::hint::black_box::h8904a2b47f70eb40(&var_30, &allocated_vec)
004063ff        return core::ptr::drop_in_place<alloc::vec::Vec<u8>>::hd93f860cc4a31edd(
004063ff            &var_30)
```

Where's the thread entry point?

```
int64_t ldr::main::h1c9a4f32d473549b()
```

```
int64_t ldr::main::h1c9a4f32d473549b()

4883ec78            sub     rsp, 0x78
488d4c2428          lea     rcx, [rsp+0x28 {var_50}]
e812beffff          call    std::thread::spawn::h062b21a91c7551cf
488d4c2428          lea     rcx, [rsp+0x28 {var_50}]
e838310000          call    core::ptr::drop_in_place...d::JoinHandle<()>>::h8ac50b7e8ebd9b70
b923000000          mov     ecx, 0x23
e80e510000          call    core::time::Duration::from_secs::h46904a934c0d6095
4889c1              mov     rcx, rax
e836380700          call    std::thread::sleep::h30cbc7896173a056
488d4c2460          lea     rcx, [rsp+0x60 {allocated_vec}]
488d159a4dee02      lea     rdx, [rel data_32eb170]
41b8480e2901        mov     r8d, 0x1290e48
e8ff500000          call    alloc::slice::<impl [T]>::to_vec::hca02de8bb68fb008
488d4c2448          lea     rcx, [rsp+0x48 {var_30}]
488d542460          lea     rdx, [rsp+0x60 {allocated_vec}]
e820510000          call    core::hint::black_box::h8904a2b47f70eb40
488d4c2448          lea     rcx, [rsp+0x48 {var_30}]
e816c60400          call    core::ptr::drop_in_place<alloc::vec::Vec<u8>>::hd93f860cc4a31edd
90                  nop
4883c478            add     rsp, 0x78
c3                  retn    {__return_addr}
```

Where's the thread entry point?

```
int64_t* std::thread::spawn::h062b21a91c7551cf(int64_t* arg1)

00402205  4889842480000000   mov     qword [rsp+0x80 {var_28}], rax
0040220d  488b0dcc2e0f00     mov     rcx, qword [rel _.rdata]  {0x0}
00402214  488b05cd2e0f00     mov     rax, qword [rel data_4f50e8]  {0x0}
0040221b  48894c2460         mov     qword [rsp+0x60 {var_48}], rcx  {0x0}
00402220  4889442468         mov     qword [rsp+0x68 {var_40}], rax  {0x0}
00402225  c684248800000000   mov     byte [rsp+0x88 {var_20}], 0x0
0040222d  488d4c2440         lea     rcx, [rsp+0x40 {var_68}]
00402232  488d542460         lea     rdx, [rsp+0x60 {var_48}]
00402237  e844000000         call    std::thread::Builder::spawn_unchecked::hfddd97d24214fbac
```

Going from the bottom up: Finding `CreateThread`

Let's get as close as possible to the point where the thread is actually created.

- This is a Windows binary that uses the Rust standard library.
- On Windows, much of the Rust standard library is implemented using the Win32 API.
- `std::thread::spawn` likely uses the Win32 `CreateThread` function as part of its implementation.

Microsoft Win32 API docs: `CreateThread`

> Creates a thread to execute within the virtual address space of the calling process.

```
HANDLE CreateThread(
  [in, optional]  LPSECURITY_ATTRIBUTES   lpThreadAttributes,
  [in]            SIZE_T                  dwStackSize,
  [in]            LPTHREAD_START_ROUTINE  lpStartAddress,
  [in, optional]  __drv_aliasesMem LPVOID lpParameter,
  [in]            DWORD                   dwCreationFlags,
  [out, optional] LPDWORD                 lpThreadId
);
```

Using `CreateThread` to find a thread entry point

To use `CreateThread` to find our thread entry:

1. Let's find whether this binary imports the `CreateThread` function.
2. Let's find where the `CreateThread` imported function is called.
3. Let's find what arguments `CreateThread` is called with. One of those arguments, `lpStartAddress`, is the thread entry point.

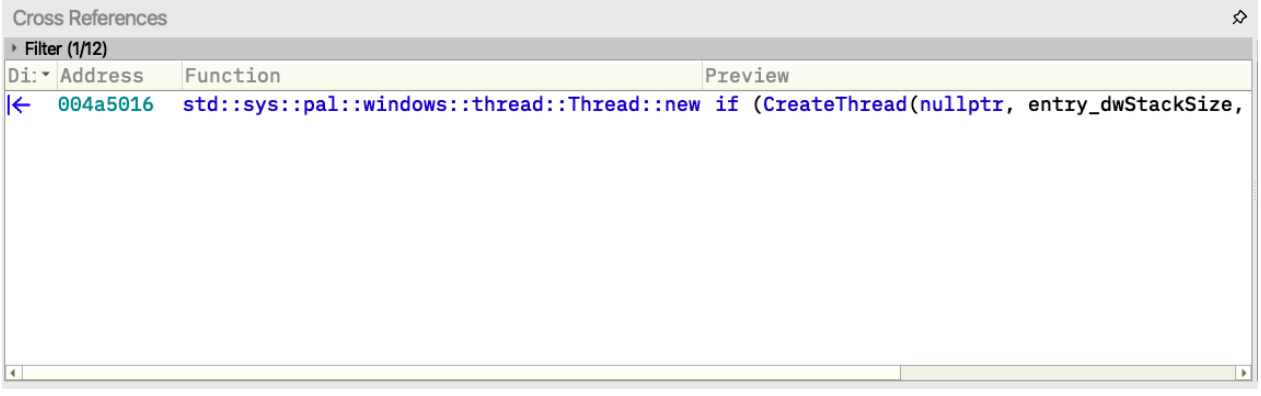
Microsoft Win32 API docs: `CreateThread`

> Creates a thread to execute within the virtual address space of the calling process.

```
HANDLE CreateThread(
  [in, optional]  LPSECURITY_ATTRIBUTES   lpThreadAttributes,
  [in]            SIZE_T                  dwStackSize,
  [in]            LPTHREAD_START_ROUTINE  lpStartAddress,
  [in, optional]  __drv_aliasesMem LPVOID lpParameter,
  [in]            DWORD                   dwCreationFlags,
  [out, optional] LPDWORD                 lpThreadId
);
```

> `[in] lpStartAddress`
> A pointer to the application-defined function to be executed by the thread. This pointer represents the starting address of the thread. For more information on the thread function, see ThreadProc.

`CreateThread` cross references



The single `CreateThread` call site

```
int64_t std::sys::pal::windows::thread::Thread::new::hfd29bd5d0469d5b4()

004a4fb0    int64_t std::sys::pal::windows::thread::Thread::new::hfd29bd5d0469d5b4()

004a4fcf        __rust_no_alloc_shim_is_unstable
004a4fdc        int64_t* lpParameter = __rust_alloc()
004a4fdc
004a4fe4        if (lpParameter == 0) {
004a508b            alloc::alloc::handle_alloc_error::hf701b546dd8b2dfe()
004a508b            noreturn
004a4fe4        }
004a4fe4
004a4fed        int64_t entry_rdx
004a4fed        *lpParameter = entry_rdx
004a4ff0        int64_t entry_r8
004a4ff0        lpParameter[1] = entry_r8
004a5021        uint64_t entry_dwStackSize
004a5021
004a5021        if (CreateThread(lpThreadAttributes: nullptr, dwStackSize: entry_dwStackSize,
004a5021                lpStartAddress:
004a5021                    std::sys::pal::windows::...::new::thread_start::h08030da9d23ee4f8,
004a5021                lpParameter, dwCreationFlags: STACK_SIZE_PARAM_IS_A_RESERVATION,
004a5021                lpThreadId: nullptr) != 0)
004a5080            return 0
004a5080
```

`CreateThread`'s `lpStartAddress` entry point

```
int64_t std::sys::pal::windows::thread::Thread::new::hfd29bd5d0469d5b4()

004a4fb0    int64_t std::sys::pal::windows::thread::Thread::new::hfd29bd5d0469d5b4()

004a4fcf        __rust_no_alloc_shim_is_unstable
004a4fdc        int64_t* lpParameter = __rust_alloc()
004a4fdc
004a4fe4        if (lpParameter == 0) {
004a508b            alloc::alloc::handle_alloc_error::hf701b546dd8b2dfe()
004a508b            noreturn
004a4fe4        }
004a4fe4
004a4fed        int64_t entry_rdx
004a4fed        *lpParameter = entry_rdx
004a4ff0        int64_t entry_r8
004a4ff0        lpParameter[1] = entry_r8
004a5021        uint64_t entry_dwStackSize
004a5021
004a5021        if (CreateThread(lpThreadAttributes: nullptr, dwStackSize: entry_dwStackSize,
004a5021                lpStartAddress:
004a5021                    std::sys::pal::windows::...::new::thread_start::h08030da9d23ee4f8,
004a5021                lpParameter, dwCreationFlags: STACK_SIZE_PARAM_IS_A_RESERVATION,
004a5021                lpThreadId: nullptr) != 0)
004a5080            return 0
004a5080
```

Inside `Thread::new::thread_start`

🤔 There's still nothing here that looks like:

- Reading a payload
- Mapping the payload into memory

```
int64_t std::sys::pal::windows::thread::Thread::new::thread_start::h08030da9d23ee4f8()

004a50f0    int64_t std::sys::pal::windows::thread::Thread::new::thread_start::h08030da9d23ee4f8()

004a5100        uint32_t StackSizeInBytes = 0x5000
004a510b        SetThreadStackGuarantee(&StackSizeInBytes)
004a5113        int64_t* entry_rcx
004a5113        void* rbx = entry_rcx[1]
004a511a        (*(rbx + 0x18))(*entry_rcx)
004a511a
004a5124        if (*(rbx + 8) != 0) {
004a5126            *(rbx + 0x10)
004a512d            __rust_dealloc()
004a5124        }
004a5124
004a5140        __rust_dealloc()
004a514f        return 0
```

An indirect call

```
int64_t std::sys::pal::windows::thread::Thread::new::thread_start::h08030da9d23ee4f8()

004a50f0    int64_t std::sys::pal::windows::thread::Thread::new::thread_start::h08030da9d23ee4f8()

004a5100        uint32_t StackSizeInBytes = 0x5000
004a510b        SetThreadStackGuarantee(&StackSizeInBytes)
004a5113        int64_t* entry_rcx
004a5113        void* rbx = entry_rcx[1]
004a511a        (*(rbx + 0x18))(*entry_rcx)
004a511a
004a5124        if (*(rbx + 8) != 0) {
004a5126            *(rbx + 0x10)
004a512d            __rust_dealloc()
004a5124        }
004a5124
004a5140        __rust_dealloc()
004a514f        return 0
```

An indirect call

```
int64_t std::sys::pal::windows::thread::Thread::new::thread_start::h08030da9d23ee4f8()

004a50f0    int64_t
004a50f0      std::sys::pal::windows::thread::Thread::new::thread_start::h08030da9d23ee4f8
004a50f0      ()

004a50f0  55                push    rbp {__saved_rbp}
004a50f1  56                push    rsi {__saved_rsi}
004a50f2  57                push    rdi {__saved_rdi}
004a50f3  53                push    rbx {__saved_rbx}
004a50f4  4883ec28          sub     rsp, 0x28
004a50f8  488d6c2420        lea     rbp, [rsp+0x20 {var_28}]
004a50fd  4889ce            mov     rsi, rcx
004a5100  c7450400500000    mov     dword [rbp+0x4 {StackSizeInBytes}], 0x5000
004a5107  488d4d04          lea     rcx, [rbp+0x4 {StackSizeInBytes}]
004a510b  e8f8c20400        call    SetThreadStackGuarantee
004a5110  488b3e            mov     rdi, qword [rsi]
004a5113  488b5e08          mov     rbx, qword [rsi+0x8]
004a5117  4889f9            mov     rcx, rdi
004a511a  ff5318            call    qword [rbx+0x18]
004a511d  488b5308          mov     rdx, qword [rbx+0x8]
004a5121  4885d2            test    rdx, rdx
004a5124  740c              je      0x4a5132
```

Where does `entry_rcx` come from?

```
int64_t std::sys::pal::windows::thread::Thread::new::thread_start::h08030da9d23ee4f8()

004a50f0    int64_t std::sys::pal::windows::thread::Thread::new::thread_start::h08030da9d23ee4f8()

004a5100        uint32_t StackSizeInBytes = 0x5000
004a510b        SetThreadStackGuarantee(&StackSizeInBytes)
004a5113        int64_t* entry_rcx
004a5113        void* rbx = entry_rcx[1]
004a511a        (*(rbx + 0x18))(*entry_rcx)
004a511a
004a5124        if (*(rbx + 8) != 0) {
004a5126            *(rbx + 0x10)
004a512d            __rust_dealloc()
004a5124        }
004a5124
004a5140        __rust_dealloc()
004a514f        return 0
```

Where does `entry_rcx` come from?

```
int64_t std::sys::pal::windows::thread::Thread::new::thread_start::h08030da9d23ee4f8()

004a50f0    int64_t std::sys::pal::windows::thread::Thread::new::thread_start::h08030da9d23ee4f8()

004a5100        uint32_t StackSizeInBytes = 0x5000
004a510b        SetThreadStackGuarantee(&StackSizeInBytes)
004a5113        int64_t* entry_rcx
004a5113        void* rbx = entry_rcx[1]
004a511a        (*(rbx + 0x18))(*entry_rcx)
004a511a
004a5124        if (*(rbx + 8) != 0) {
004a5126            *(rbx + 0x10)
004a512d            __rust_dealloc()
004a5124        }
004a5124
004a5140        __rust_dealloc()
004a514f        return 0
```

Where does `entry_rcx` come from?

This value doesn't come from anywhere within the function.

- It's just the value in `rcx` at the beginning of the function.

```
int64_t std::sys::pal::windows::thread::Thread::new::thread_start::h08030da9d23ee4f8()
004a50f3  53              push    rbx {__saved_rbx}
004a50f4  4883ec28        sub     rsp, 0x28
004a50f8  488d6c2420      lea     rbp, [rsp+0x20 {var_28}]
004a50fd  4889ce          mov     rsi, rcx
004a5100  c7450400500000  mov     dword [rbp+0x4 {StackSizeInBytes}], 0x5000
004a5107  488d4d04        lea     rcx, [rbp+0x4 {StackSizeInBytes}]
004a510b  e8f8c20400      call    SetThreadStackGuarantee
004a5110  488b3e          mov     rdi, qword [rsi]
004a5113  488b5e08        mov     rbx, qword [rsi+0x8]
004a5117  4889f9          mov     rcx, rdi
004a511a  ff5318          call    qword [rbx+0x18]
004a511d  488b5308        mov     rdx, qword [rbx+0x8]
004a5121  4885d2          test    rdx, rdx
```

The Windows x64 ABI and calling convention

Microsoft developer docs: Windows x64 calling convention > Parameter passing

Parameter type	leftmost	second	third	fourth	fifth and higher
integer	RCX	RDX	R8	R9	stack
Aggregates (8, 16, 32, or 64 bits) and `__m64`	RCX	RDX	R8	R9	stack
Other aggregates, as pointers	RCX	RDX	R8	R9	stack

Note

We can trust here that the Windows x64 calling convention is being used, because this is not Rust code calling Rust code. This is a case where Rust code must interoperate with C code (FFI)

It's Rust code (`thread::Thread::new`) -> C code (Win32 `CreateThread`) -> Rust code (`Thread::new::thread_start`)

How `Thread::new::thread_start` was called: via `CreateThread`

```
004a5021    if (CreateThread(lpThreadAttributes: nullptr, dwStackSize: entry_dwStackSize,
004a5021            lpStartAddress:
004a5021                std::sys::pal::windows::...::new::thread_start::h08030da9d23ee4f8,
004a5021            lpParameter, dwCreationFlags: STACK_SIZE_PARAM_IS_A_RESERVATION,
004a5021            lpThreadId: nullptr) != 0)
004a5080        return 0
004a5080
```

```
HANDLE CreateThread(
  [in, optional]  LPSECURITY_ATTRIBUTES   lpThreadAttributes,
  [in]            SIZE_T                  dwStackSize,
  [in]            LPTHREAD_START_ROUTINE  lpStartAddress,
  [in, optional]  __drv_aliasesMem LPVOID lpParameter,
  [in]            DWORD                   dwCreationFlags,
  [out, optional] LPDWORD                 lpThreadId
);
```

> `[in] lpStartAddress`
>
> A pointer to the application-defined function to be executed by the thread. This pointer represents the starting address of the thread.

`CreateThread`'s `lpStartAddress` thread entry point parameter

Microsoft developer docs: `CreateThread`

> `[in] lpStartAddress`
>
> A pointer to the application-defined function to be executed by the thread. This pointer represents the starting address of the thread. For more information on the thread function, see ThreadProc.

> *ThreadProc callback function*
>
> An application-defined function that serves as the starting address for a thread. Specify this address when calling the CreateThread, CreateRemoteThread, or CreateRemoteThreadEx function.

```
DWORD WINAPI ThreadProc(
  _In_ LPVOID lpParameter
);
```

> `[in] lpParameter`
>
> The thread data passed to the function using the *lpParameter* parameter of the CreateThread, CreateRemoteThread, or CreateRemoteThreadEx function.

The thread entry point function's `lpParameter`

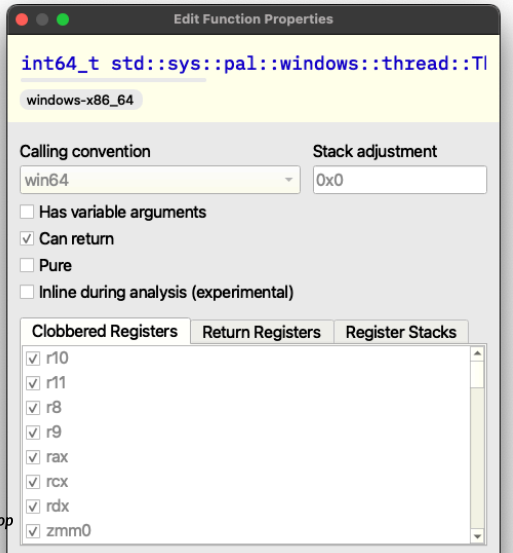
```
HANDLE CreateThread(
  [in, optional]  LPSECURITY_ATTRIBUTES   lpThreadAttributes,
  [in]            SIZE_T                  dwStackSize,
  [in]            LPTHREAD_START_ROUTINE  lpStartAddress,
  [in, optional]  __drv_aliasesMem LPVOID lpParameter,
  [in]            DWORD                   dwCreationFlags,
  [out, optional] LPDWORD                 lpThreadId
);
```

```
DWORD WINAPI ThreadProc(
  _In_ LPVOID lpParameter
);
```

```
DWORD WINAPI std::sys::pal::windows::thread::Thread::new::thread_start(
    LPVOID lpParameter
);
```

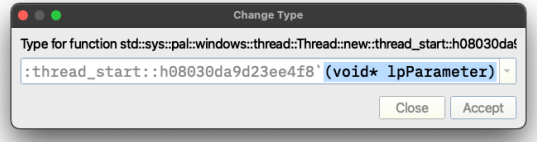
Setting `rcx` for `Thread::new::thread_start`

Ensure the Calling Convention is `win64`, via *right-click menu > Edit Function Properties* (or `Shift+E`).



Add 1 argument to `Thread::new::thread_start`, via *right-click menu > Change Type* (or press `Y`).

```
(void* lpParameter)
```



Binary Ninja will place this first argument in `rcx`, according to the Windows x64 calling convention.

Parameter type	1st	2nd	3rd	4th	5th
Other aggregates, as pointers	RCX	RDX	R8	R9	stack

Setting `rcx` for `Thread::new::thread_start`

```
int64_t std::sys::pal::windows::thread::Thread::new::thread_start::h08030da9d23ee4f8(void* lpParam

004a50f0    int64_t std::sys::pal::windows::thread::Thread::new::thread_start::h08030da9d23ee4f8(
004a50f0      void* lpParameter)

004a5100      uint32_t StackSizeInBytes = 0x5000
004a510b      SetThreadStackGuarantee(&StackSizeInBytes)
004a5113      void* rbx = *(lpParameter + 8)
004a511a      (*(rbx + 0x18))(*lpParameter)
004a511a
004a5124      if (*(rbx + 8) != 0) {
004a5126          *(rbx + 0x10)
004a512d          __rust_dealloc()
004a5124      }
004a5124
004a5140      __rust_dealloc()
004a514f      return 0
```

Examining `lpParameter` at the `CreateThread` call site

```
int64_t std::sys::pal::windows::thread::Thread::new::hfd29bd5d0469d5b4()

004a4fb0    int64_t std::sys::pal::windows::thread::Thread::new::hfd29bd5d0469d5b4()

004a4fcf        __rust_no_alloc_shim_is_unstable
004a4fdc        int64_t* lpParameter = __rust_alloc()
004a4fdc
004a4fe4        if (lpParameter == 0) {
004a508b            alloc::alloc::handle_alloc_error::hf701b546dd8b2dfe()
004a508b            noreturn
004a4fe4        }
004a4fe4
004a4fed        int64_t entry_rdx
004a4fed        *lpParameter = entry_rdx
004a4ff0        int64_t entry_r8
004a4ff0        lpParameter[1] = entry_r8
004a5021        uint64_t entry_dwStackSize
004a5021
004a5021        if (CreateThread(lpThreadAttributes: nullptr, dwStackSize: entry_dwStackSize,
004a5021                lpStartAddress:
004a5021                    std::sys::pal::windows::...::new::thread_start::h08030da9d23ee4f8,
004a5021                lpParameter, dwCreationFlags: STACK_SIZE_PARAM_IS_A_RESERVATION,
004a5021                lpThreadId: nullptr) != 0)
004a5080            return 0
004a5080
```

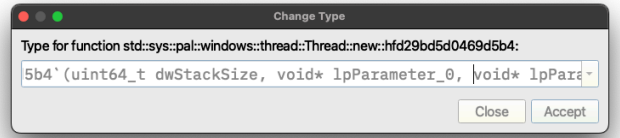
Tracing where `lpParameter` comes from

```
int64_t std::sys::pal::windows::thread::Thread::new::hfd29bd5d0469d5b4()

004a4fb0    int64_t std::sys::pal::windows::thread::Thread::new::hfd29bd5d0469d5b4()

004a4fcf        __rust_no_alloc_shim_is_unstable
004a4fdc        int64_t* lpParameter = __rust_alloc()
004a4fdc
004a4fe4        if (lpParameter == 0) {
004a508b            alloc::alloc::handle_alloc_error::hf701b546dd8b2dfe()
004a508b            noreturn
004a4fe4        }
004a4fe4
004a4fed        int64_t entry_rdx
004a4fed        *lpParameter = entry_rdx
004a4ff0        int64_t entry_r8
004a4ff0        lpParameter[1] = entry_r8
004a5021        uint64_t entry_dwStackSize
004a5021
004a5021        if (CreateThread(lpThreadAttributes: nullptr, dwStackSize: entry_dwStackSize,
004a5021                lpStartAddress:
004a5021                    std::sys::pal::windows::...::new::thread_start::h08030da9d23ee4f8,
004a5021                lpParameter, dwCreationFlags: STACK_SIZE_PARAM_IS_A_RESERVATION,
004a5021                lpThreadId: nullptr) != 0)
004a5080            return 0
004a5080
```

Tracing where `lpParameter` comes from

Add 3 arguments to `thread::Thread::new`, via *right-click menu > Change Type* (or press `Y`).

```
(uint64_t dwStackSize, void* lpParameter_0, void* lpParameter_1)
```



Binary Ninja will place the arguments in `rcx`, `rdx`, and `r8`, according to the Windows x64 calling convention.

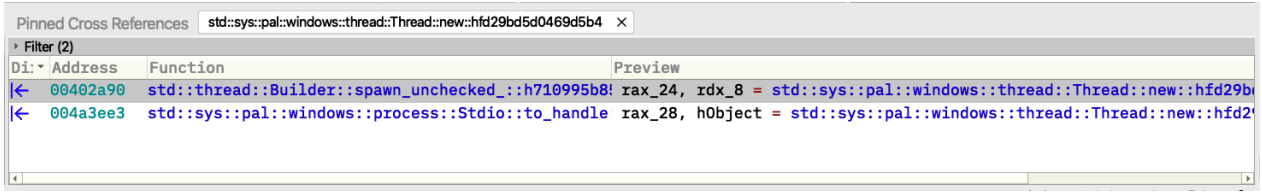
Parameter type	1st	2nd	3rd	4th	5th
Integer	RCX	RDX	R8	R9	stack
Other aggregates, as pointers	RCX	RDX	R8	R9	stack

Tracing where `lpParameter` comes from

```
int64_t std::sys::pal::windows::thread::Thread::new::hfd29bd5d0469d5b4(uint64_t dwStackSize, void* l

004a4fb0    int64_t std::sys::pal::windows::thread::Thread::new::hfd29bd5d0469d5b4(
004a4fb0      uint64_t dwStackSize, void* lpParameter_0, void* lpParameter_1)

004a4fcf        __rust_no_alloc_shim_is_unstable
004a4fdc        void** lpParameter = __rust_alloc()
004a4fdc
004a4fe4        if (lpParameter == 0) {
004a508b            alloc::alloc::handle_alloc_error::hf701b546dd8b2dfe()
004a508b            noreturn
004a4fe4        }
004a4fe4
004a4fed        *lpParameter = lpParameter_0
004a4ff0        lpParameter[1] = lpParameter_1
004a4ff0
004a5021        if (CreateThread(lpThreadAttributes: nullptr, dwStackSize,
004a5021                lpStartAddress:
004a5021                    std::sys::pal::windows::...::new::thread_start::h08030da9d23ee4f8,
004a5021                lpParameter, dwCreationFlags: STACK_SIZE_PARAM_IS_A_RESERVATION,
004a5021                lpThreadId: nullptr) != 0)
004a5080            return 0
```

Tracing where the `lpParameter` args come from

Examining the cross-references to `std::sys::pal::windows::thread::Thread::new`:



The call site at `std::thread::Builder::spawn_unchecked_`

```
int64_t* std::thread::Builder::spawn_unchecked_::h710995b852c21546(int64_t* arg1, int64_t* arg2,

0040297e        char var_9e_1 = 0
0040298e        int64_t* var_f8 = rax_19
004029ae        int128_t var_108 = var_1e0.o
004029b6        int128_t var_118 = var_1f0.o
004029be        int128_t var_128 = rax_11.o
004029d0        void* lpParameter_0 = alloc::alloc::exchange_malloc::ha2549f6bdd3548c0(0x38, 8)
00402a31        memcpy(dest: lpParameter_0, src: &var_128, n: 0x38)
00402a90        int64_t rax_24
00402a90        int64_t rdx_8
00402a90        rax_24, rdx_8 = std::sys::pal::windows::thread::Thread::new::hfd29bd5d0469d5b4(
00402a90            dwStackSize, lpParameter_0, lpParameter_1: &data_4f5170)
00402a90
00402ac4        if (rax_24 != 0) {
00402b87            int64_t var_c8_1 = 1
00402bab            arg1[1] = rdx_8
```

Finally! Some concrete data

```
int64_t* std::thread::Builder::spawn_unchecked_::h710995b852c21546(int64_t* arg1, int64_t* arg2,

0040297e        char var_9e_1 = 0
0040298e        int64_t* var_f8 = rax_19
004029ae        int128_t var_108 = var_1e0.o
004029b6        int128_t var_118 = var_1f0.o
004029be        int128_t var_128 = rax_11.o
004029d0        void* lpParameter_0 = alloc::alloc::exchange_malloc::ha2549f6bdd3548c0(0x38, 8)
00402a31        memcpy(dest: lpParameter_0, src: &var_128, n: 0x38)
00402a90        int64_t rax_24
00402a90        int64_t rdx_8
00402a90        rax_24, rdx_8 = std::sys::pal::windows::thread::Thread::new::hfd29bd5d0469d5b4(
00402a90            dwStackSize, lpParameter_0, lpParameter_1: &data_4f5170)
00402a90
00402ac4        if (rax_24 != 0) {
00402b87            int64_t var_c8_1 = 1
00402bab            arg1[1] = rdx_8
```

But what does this data mean?

```
0x4f5158 .rdata {0x4f5000-0x45a5d20} Read-only data

004f5158  data_4f5158:
004f5158                           06 51 4f 00 00 00 00 00          .QO.....
004f5160  4d 00 00 00 00 00 00 00-de 02 00 00 1d 00 00 00  M...............
004f5170  data_4f5170:
004f5170  c0 8f 40 00 00 00 00 00-38 00 00 00 00 00 00 00  ..@.....8.......
004f5180  08 00 00 00 00 00 00 00-70 8e 40 00 00 00 00 00  ........p.@.....

004f5190  char const data_4f5190[0x44] = "RUST_MIN_STACKfatal runtime error: something here is"
004f5190      " badly broken!\n", 0

004f51d4              00 00 00 00                                ....
004f51d8  data_4f51d8:
004f51d8                           9e 51 4f 00 00 00 00 00          .QO.....
004f51e0  35 00 00 00 00 00 00 00-66 61 74 61 6c 20 72 75  5.......fatal ru
004f51f0  6e 74 69 6d 65 20 65 72-72 6f 72 3a 20 74 68 72  ntime error: thr
```

Traits in Rust

- In Rust, a *trait* is a function interface that a type must adhere to. It is defined as a collection of functions that a type must implement.
  - In Golang, Java, or C#: you may know this as an *interface*.
  - In C++: you may know this as an *abstract base class*.
- In Rust, a *trait object* is a pointer to:
  - Some concrete object
  - Some information about a trait that the object implements.
- The compiler doesn't know what the actual type of this object is, only that the type exposes a certain function interface.

&dyn *Trait*	
00	concrete_object: **ConcreteType*
08	vtable: **<impl Trait for ConcreteType>::_vtable*

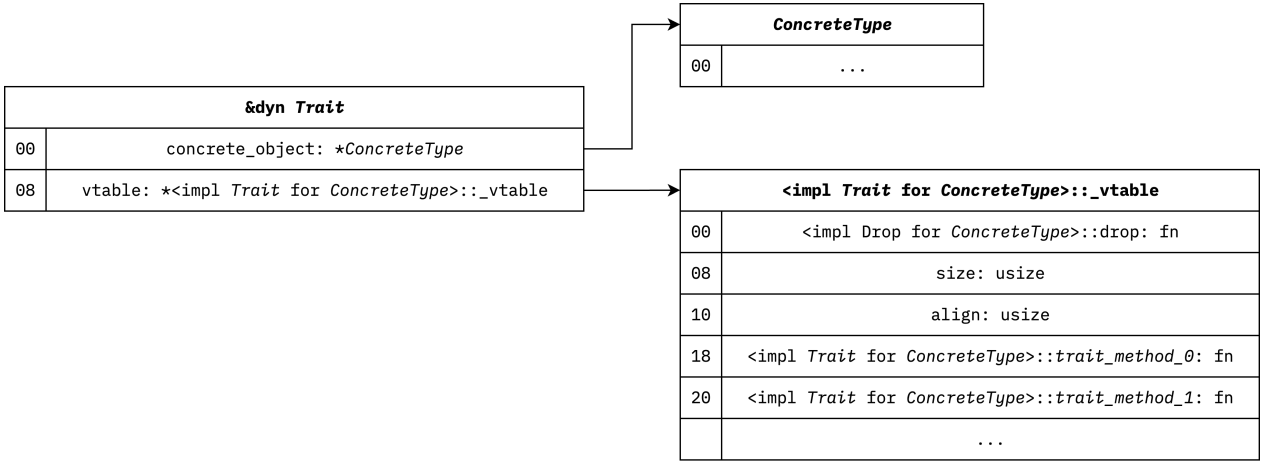
Virtual function tables (vtables) in Rust

Trait vtables are statically embedded in the binary and always have a fixed layout:

- A function pointer to the *destructor* for the concrete type that implements the trait.
- The *size* of the concrete type in bytes.
- The *alignment* of the concrete type in bytes.
- An array of function pointers, pointing to the implementation, for the concrete type, for each required *trait method*.

<impl *Trait* for *ConcreteType*>::_vtable	
00	<impl Drop for *ConcreteType*>::drop: fn
08	size: usize
10	align: usize
18	<impl *Trait* for *ConcreteType*>::*trait_method_0*: fn
20	<impl *Trait* for *ConcreteType*>::*trait_method_1*: fn
	...

Traits & vtables, together



A C-like representation of a vtable

```
struct `<impl Trait for ConcreteType>::_vtable` __packed
{
    void* (* `<impl Drop for ConcreteType>::drop`)(ConcreteType*);
    uint64_t size;
    uint64_t align;
    void* (* `<impl Trait for ConcreteType>::trait_method_0`)(void*);
    void* (* `<impl Trait for ConcreteType>::trait_method_1`)(void*);
    [...]
};
```

<impl *Trait* for *ConcreteType*>::_vtable	
00	<impl Drop for *ConcreteType*>::drop: fn
08	size: usize
10	align: usize
18	<impl *Trait* for *ConcreteType*>::*trait_method_0*: fn
20	<impl *Trait* for *ConcreteType*>::*trait_method_1*: fn
	...

Closures in Rust

In Rust, closures (i.e. anonymous functions) are implemented as types that implement the `std::fn::FnOnce` trait.

This trait only has one required method, that a closure type must implement:

```
pub trait FnOnce<Args>
{
    // Required method
    extern "rust-call" fn call_once(self, args: Args) -> Self::Output;
}
```

<impl std::ops::FnOnce for *ClosureType*>::_vtable	
00	<impl Drop for *ClosureType*>::drop: fn
08	size: usize
10	align: usize
18	<impl std::ops::FnOnce for *ClosureType*>::call_once: fn

Annotating our vtable

<impl std::ops::FnOnce for *ClosureType*>::_vtable	
00	<impl Drop for *ClosureType*>::drop: fn
08	size: usize
10	align: usize
18	<impl std::ops::FnOnce for *ClosureType*>::call_once: fn

```
0x4f5170 .rdata {0x4f5000-0x45a5d20} Read-only data

004f5160  4d 00 00 00 00 00 00 00  M.......
004f5168  de 02 00 00 1d 00 00 00  ........
004f5170  data_4f5170:
004f5170  c0 8f 40 00 00 00 00 00  ..@.....
004f5178  38 00 00 00 00 00 00 00  8.......
004f5180  08 00 00 00 00 00 00 00  ........
004f5188  70 8e 40 00 00 00 00 00  p.@.....

004f5190  char const data_4f5190[0x44] = "RUST_MIN_STAC"
004f5190      "Kfatal runtime error: something here is "
004f5190      "badly broken!\n", 0
```

Setting up a vtable type

Right-click in Types window > *Create Types from C Source* > Enter C-like type definitions

```
struct ClosureType {}; // This is just a placeholder for now

struct `<impl std::ops::FnOnce for ClosureType>::_vtable` __packed
{
    void* (* `<impl Drop for ClosureType>::drop`)(ClosureType*);
    uint64_t size;
    uint64_t align;
    void* (* `<impl std::ops::FnOnce for ClosureType>::call_once`)(ClosureType*, void*);
};
```

Annotating our vtable

```
struct ClosureType {}; // This is just a placeholder for now

struct `<impl std::ops::FnOnce for ClosureType>::_vtable` __packed
{
    void* (* `<impl Drop for ClosureType>::drop`)(ClosureType*);
    uint64_t size;
    uint64_t align;
    void* (* `<impl std::ops::FnOnce for ClosureType>::call_once`)(ClosureType*, void*);
};
```

```
0x4f5158 .rdata {0x4f5000-0x45a5d20} Read-only data

004f5158                         06 51 4f 00 00 00 00 00          .QO.....
004f5160 4d 00 00 00 00 00 00 00-de 02 00 00 1d 00 00 00  M...............

004f5170 struct <impl std::ops::FnOnce for ClosureType>::_vtable data_4f5170 =
004f5170 {
004f5170     void* (* <impl Drop for ClosureType>::drop)(struct ClosureType*) = core::ptr::drop_in_place<std::t
004f5178     uint64_t size = 0x38
004f5180     uint64_t align = 0x8
004f5188     void* (* <impl std::ops::FnOnce for ClosureType>::call_once)(struct ClosureType*, void*) = core::o
004f5190 }
004f5190 char const data_4f5190[0x44] = "RUST_MIN_STACKfatal runtime error: something here is badly broken!\n",
```

Going back up to where the vtable is used

```
int64_t* std::thread::Builder::spawn_unchecked_::h710995b852c21546(int64_t* arg1, int64_t* arg2, int6

0040297e    char var_9e_1 = 0
0040298e    int64_t* var_f8 = rax_19
004029ae    int128_t var_108 = var_1e0.o
004029b6    int128_t var_118 = var_1f0.o
004029be    int128_t var_128 = rax_11.o
004029d0    void* lpParameter_0 = alloc::alloc::exchange_malloc::ha2549f6bdd3548c0(0x38, 8)
00402a31    memcpy(dest: lpParameter_0, src: &var_128, n: 0x38)
00402a90    int64_t rax_24
00402a90    int64_t rdx_8
00402a90    rax_24, rdx_8 = std::sys::pal::windows::thread::Thread::new::hfd29bd5d0469d5b4(
00402a90        dwStackSize, lpParameter_0,
00402a90        lpParameter_1: &_<impl std::ops::FnOnce for ClosureType>::_vtable)
00402a90
```

Re-typing the argument for `thread::Thread::new`

```
int64_t std::sys::pal::windows::thread::Thread::new::hfd29bd5d0469d5b4(uint64_t dwStackSize, void* lp

004a4fb0    int64_t std::sys::pal::windows::thread::Thread::new::hfd29bd5d0469d5b4(
004a4fb0      uint64_t dwStackSize, void* lpParameter_0,
004a4fb0      struct <impl std::ops::FnOnce for ClosureType>::_vtable* lpParameter_1)

004a4fcf        __rust_no_alloc_shim_is_unstable
004a4fdc        void** lpParameter = __rust_alloc()
004a4fdc
004a4fe4        if (lpParameter == 0) {
004a508b            alloc::alloc::handle_alloc_error::hf701b546dd8b2dfe()
004a508b            noreturn
004a4fe4        }
004a4fe4
004a4fed        *lpParameter = lpParameter_0
004a4ff0        lpParameter[1] = lpParameter_1
004a4ff0
004a5021        if (CreateThread(lpThreadAttributes: nullptr, dwStackSize,
004a5021                lpStartAddress:
004a5021                    std::sys::pal::windows::...::new::thread_start::h08030da9d23ee4f8,
004a5021                lpParameter, dwCreationFlags: STACK_SIZE_PARAM_IS_A_RESERVATION,
004a5021                lpThreadId: nullptr) != 0)
004a5080            return 0
004a5080
```

What's in `lpParameter`?

We have arguments `lpParameter_0` and `lpParameter_1`, but only the local variable `lpParameter` is passed into `CreateThread` (and our `thread_start` thread entry point)

```
int64_t std::sys::pal::windows::thread::Thread::new::hfd29bd5d0469d5b4(uint64_t dwStackSize, void* lp

004a4fb0    int64_t std::sys::pal::windows::thread::Thread::new::hfd29bd5d0469d5b4(
004a4fb0      uint64_t dwStackSize, void* lpParameter_0,
004a4fb0      struct <impl std::ops::FnOnce for ClosureType>::_vtable* lpParameter_1)

004a4fcf      __rust_no_alloc_shim_is_unstable
004a4fdc      void** lpParameter = __rust_alloc()
004a4fdc
004a4fe4      if (lpParameter == 0) {
004a508b          alloc::alloc::handle_alloc_error::hf701b546dd8b2dfe()
004a508b          noreturn
004a4fe4      }
004a4fe4
004a4fed      *lpParameter = lpParameter_0
004a4ff0      lpParameter[1] = lpParameter_1
004a4ff0
004a5021      if (CreateThread(lpThreadAttributes: nullptr, dwStackSize,
004a5021              lpStartAddress:
004a5021                  std::sys::pal::windows::...::new::thread_start::h08030da9d23ee4f8,
004a5021              lpParameter, dwCreationFlags: STACK_SIZE_PARAM_IS_A_RESERVATION,
004a5021              lpThreadId: nullptr) != 0)
004a5080          return 0
```

Examining `__rust_alloc`

```
int64_t std::sys::pal::windows::thread::Thread::new::hfd29bd5d0469d5b4(uint64_t dwStackSize, void* lp

004a4fb0    int64_t std::sys::pal::windows::thread::Thread::new::hfd29bd5d0469d5b4(
004a4fb0      uint64_t dwStackSize, void* lpParameter_0,
004a4fb0      struct <impl std::ops::FnOnce for ClosureType>::_vtable* lpParameter_1)

004a4fcf        __rust_no_alloc_shim_is_unstable
004a4fdc        void** lpParameter = __rust_alloc()
004a4fdc
004a4fe4        if (lpParameter == 0) {
004a508b            alloc::alloc::handle_alloc_error::hf701b546dd8b2dfe()
004a508b            noreturn
004a4fe4        }
004a4fe4
004a4fed        *lpParameter = lpParameter_0
004a4ff0        lpParameter[1] = lpParameter_1
004a4ff0
004a5021        if (CreateThread(lpThreadAttributes: nullptr, dwStackSize,
004a5021                lpStartAddress:
004a5021                    std::sys::pal::windows::...::new::thread_start::h08030da9d23ee4f8,
004a5021                lpParameter, dwCreationFlags: STACK_SIZE_PARAM_IS_A_RESERVATION,
004a5021                lpThreadId: nullptr) != 0)
004a5080            return 0
```

Examining `__rust_alloc`

Rust standard library docs: `__rust_alloc`

```
unsafe fn __rust_alloc(size: usize, align: usize) -> *mut u8
```

Re-type `__rust_alloc` to:

```
void* __rust_alloc(uint64_t size, uint64_t align)
```

```
int64_t std::sys::pal::windows::thread::Thread::new::hfd29bd5d0469d5b4(uint64_t dwStack
004a4fc2  4989d6            mov     r14, rdx
004a4fc5  4889cf            mov     rdi, rcx
004a4fc8  488b05890c1004    mov     rax, qword [rel _.rdata$.refptr.__rust_no_alloc_sh
004a4fcf  0fb600            movzx   eax, byte [rax]  {__rust_no_alloc_shim_is_unstable
004a4fd2  b910000000        mov     ecx, 0x10
004a4fd7  ba08000000        mov     edx, 0x8
004a4fdc  e84f65f6ff        call    __rust_alloc
004a4fe1  4885c0            test    rax, rax
004a4fe4  0f8497000000      je      0x4a5081

004a4fea  4889c6            mov     rsi, rax
```

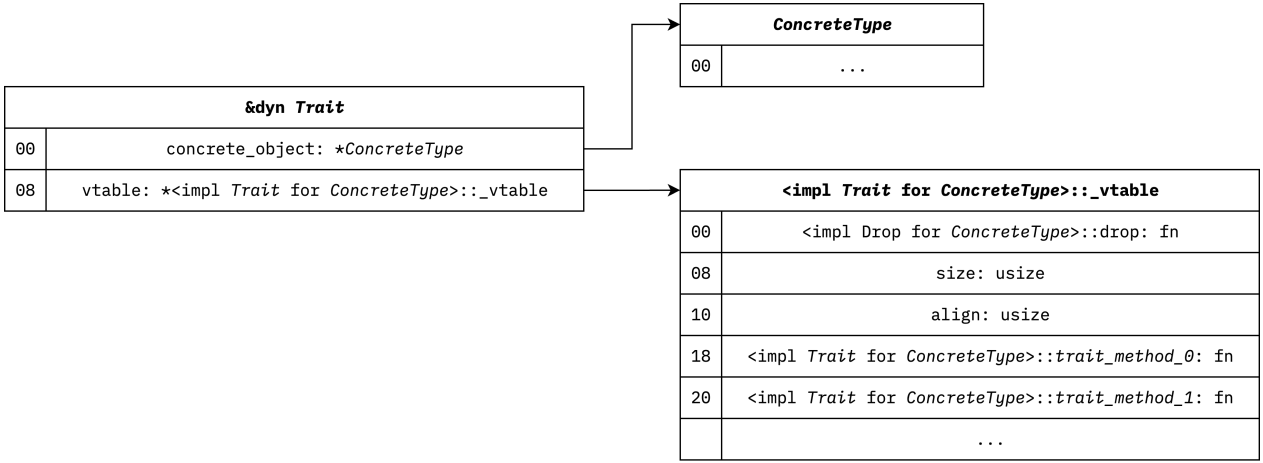
Allocating memory for `lpParameter` using `__rust_alloc`

The local variable `lpParameter` is something with a size of 0x10 bytes, and alignment of 0x8 bytes.

- The argument `lpParameter_0` is stored at offset 0x0 bytes.
- The argument `lpParameter_1` (the vtable argument) is at offset 0x8 bytes.

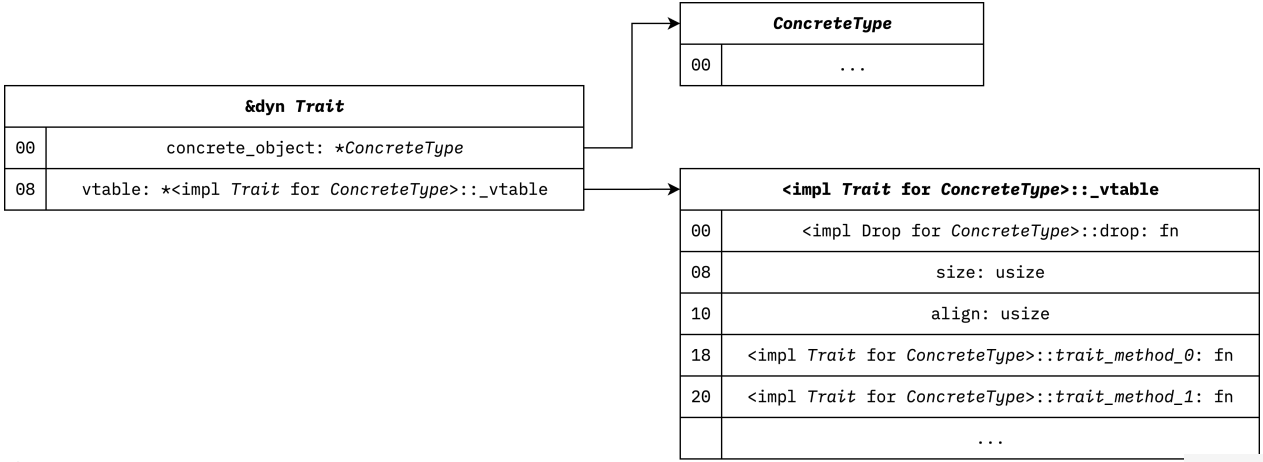
```
int64_t std::sys::pal::windows::thread::Thread::new::hfd29bd5d0469d5b4(uint64_t dwStackSize, void* lp

004a4fb0    int64_t std::sys::pal::windows::thread::Thread::new::hfd29bd5d0469d5b4(
004a4fb0      uint64_t dwStackSize, void* lpParameter_0,
004a4fb0      struct <impl std::ops::FnOnce for ClosureType>::_vtable* lpParameter_1)

004a4fcf        __rust_no_alloc_shim_is_unstable
004a4fdc        int64_t lpParameter = __rust_alloc(size: 0x10, align: 8)
004a4fdc
004a4fe4        if (lpParameter == 0) {
004a508b            alloc::alloc::handle_alloc_error::hf701b546dd8b2dfe()
004a508b            noreturn
004a4fe4        }
004a4fe4
004a4fed        *lpParameter = lpParameter_0
004a4ff0        *(lpParameter + 8) = lpParameter_1
004a4ff0
004a5021        if (CreateThread(lpThreadAttributes: nullptr, dwStackSize,
004a5021                lpStartAddress:
004a5021                    std::sys::pal::windows::...::new::thread_start::h08030da9d23ee4f8,
004a5021                lpParameter, dwCreationFlags: STACK_SIZE_PARAM_IS_A_RESERVATION,
004a5021                lpThreadId: nullptr) != 0)
004a5080            return 0
004a5080
```

Traits & vtables, together, again



Defining the trait type

```
struct `&dyn std::ops::FnOnce` __packed
{
    ClosureType* concrete_object;
    `<impl std::ops::FnOnce for ClosureType>::_vtable`* vtable;
};
```



Applying the trait type

```
int64_t std::sys::pal::windows::thread::Thread::new::hfd29bd5d0469d5b4(uint64_t dwStackSize, vo

004a4fb0    int64_t std::sys::pal::windows::thread::Thread::new::hfd29bd5d0469d5b4(
004a4fb0      uint64_t dwStackSize, void* lpParameter_0,
004a4fb0      struct <impl std::ops::FnOnce for ClosureType>::_vtable* lpParameter_1)

004a4fcf        __rust_no_alloc_shim_is_unstable
004a4fdc        struct &dyn std::ops::FnOnce* lpParameter = __rust_alloc(size: 0x10, align: 8)
004a4fdc
004a4fe4        if (lpParameter == 0) {
004a508b            alloc::alloc::handle_alloc_error::hf701b546dd8b2dfe()
004a508b            noreturn
004a4fe4        }
004a4fe4
004a4fed        lpParameter->concrete_object = lpParameter_0
004a4ff0        lpParameter->vtable = lpParameter_1
004a4ff0
004a5021        if (CreateThread(lpThreadAttributes: nullptr, dwStackSize,
004a5021                lpStartAddress:
004a5021                    std::sys::pal::windows::...::new::thread_start::h08030da9d23ee4f8,
004a5021                lpParameter, dwCreationFlags: STACK_SIZE_PARAM_IS_A_RESERVATION,
004a5021                lpThreadId: nullptr) != 0)
004a5080            return 0
004a5080
```

Applying the trait type inside the thread entry function, `thread_start`

```
int64_t std::sys::pal::windows::thread::Thread::new::thread_start::h08030da9d23ee4f8(struct &

004a50f0    int64_t
004a50f0      std::sys::pal::windows::thread::Thread::new::thread_start::h08030da9d23ee4f8(
004a50f0      struct &dyn std::ops::FnOnce* lpParameter)

004a5100        uint32_t StackSizeInBytes = 0x5000
004a510b        void* rdx = SetThreadStackGuarantee(&StackSizeInBytes)
004a5113        struct <impl std::ops::FnOnce for ClosureType>::_vtable* vtable =
004a5113            lpParameter->vtable
004a511a        vtable-><impl std::ops::FnOnce for ClosureType>::call_once(
004a511a            lpParameter->concrete_object, rdx)
004a511a
004a5124        if (vtable->size != 0) {
004a5126            vtable->align
004a512d            __rust_dealloc()
004a5124        }
004a5124
004a5140        __rust_dealloc()
004a514f        return 0
```

Finally invoking the closure's `call_once` function!

```
pub trait FnOnce<Args> { extern "rust-call" fn call_once(self, args: Args) -> Self::Output; }
```

```
int64_t std::sys::pal::windows::thread::Thread::new::thread_start::h08030da9d23ee4f8(struct &

004a50f0    int64_t
004a50f0      std::sys::pal::windows::thread::Thread::new::thread_start::h08030da9d23ee4f8(
004a50f0      struct &dyn std::ops::FnOnce* lpParameter)

004a5100        uint32_t StackSizeInBytes = 0x5000
004a510b        void* rdx = SetThreadStackGuarantee(&StackSizeInBytes)
004a5113        struct <impl std::ops::FnOnce for ClosureType>::_vtable* vtable =
004a5113            lpParameter->vtable
004a511a        vtable-><impl std::ops::FnOnce for ClosureType>::call_once(
004a511a            lpParameter->concrete_object, rdx)
004a511a
004a5124        if (vtable->size != 0) {
004a5126            vtable->align
004a512d            __rust_dealloc()
004a5124        }
004a5124
004a5140        __rust_dealloc()
004a514f        return 0
```

Back to the vtable, following the `call_once` function pointer

```
0x4f5158 .rdata {0x4f5000-0x45a5d20} Read-only data

004f5158                                                    06 51 4f 00 00 00 00 00                 .QO.....
004f5160 4d 00 00 00 00 00 00 00-de 02 00 00 1d 00 00 00    M...............

004f5170 struct <impl std::ops::FnOnce for ClosureType>::_vtable data_4f5170 =
004f5170 {
004f5170     void* (* <impl Drop for ClosureType>::drop)(struct ClosureType*) = core::ptr::drop_in_place<std::thread::Builder::spawn_unchecked_<ldr::omg,()>::{{closure}}>::h7f0f78abce4a83e3
004f5178     uint64_t size = 0x38
004f5180     uint64_t align = 0x8
004f5188     void* (* <impl std::ops::FnOnce for ClosureType>::call_once)(struct ClosureType*, void*) = core::ops::function::FnOnce::call_once{{vtable.shim}}::h3c51115acb4ffa9f
004f5190 }
```

Following the `call_once` function pointer

```
int64_t core::ops::function::FnOnce::call_once{{vtable.shim}}::h3c51115acb4ffa9f(int64_t* arg1)

00408e70    int64_t core::ops::function::FnOnce::call_once{{vtable.shim}}::h3c51115acb4ffa9f(int64_t* arg1)

00408e7e        return std::thread::Builder::sp...cked_::{{closure}}::hcc36a8b3b9fab37c(arg1)
```

Following the `call_once` function pointer

```
int64_t std::thread::Builder::spawn_unchecked_::{{closure}}::hcc36a8b3b9fab37c(int64_t* arg1)

004030b0    int64_t rcx_11 = arg1[4]
004030bc    int64_t rax_11 = arg1[5]
00403150    int64_t var_58 = arg1[2]
00403170    int64_t var_48 = rcx_11
00403180    int64_t var_40 = rax_11
0040319e    __rust_try(std::panicking::try::do_call::h5688bcf50b7b135f, &var_58)
004031a3
004031fe
0040320e
0040321e
00403227
00403283
00403283
0040328f
00403292
0040329e
004032ae
004032c7
```

```
int64_t std::panicking::try::do_call::h5688bcf50b7b135f(int64_t* arg1)

0040ae70    int64_t std::panicking::try::do_call::h5688bcf50b7b135f(int64_t* arg1)

0040ae77        int64_t var_20 = *arg1
0040ae80        int64_t var_18 = arg1[1]
0040ae89        int64_t var_10 = arg1[2]
0040ae92        int64_t var_8 = arg1[3]
0040aea6        return <core::panic::unwind_saf...ce<()>>::call_once::h6339cc4ba8b8453c(&var_20)
```

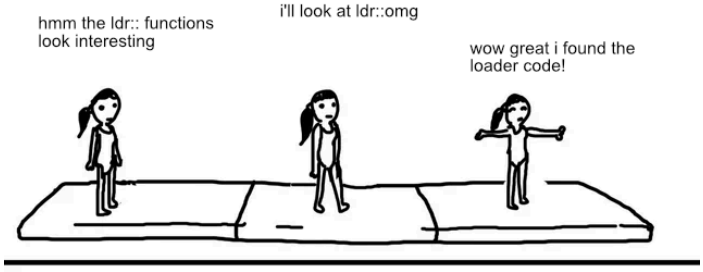
```
int64_t std::thread::Builder::spawn_unchecked_::{{closure}}::{{closure}}::he40f6656490c4b8d(int128_t* arg1)

00403490    int64_t std::thread::Builder::spawn_unchecked_::{{closure}}::{{closure}}::he40f6656490c4b8d(
00403490      int128_t* arg1)

00403494        char var_11 = 1
004034a0        int128_t var_28 = arg1[1]
004034a5        int128_t var_38 = *arg1
004034af        std::sys::backtrace::__r...in_short_backtrace::h1540133d140e987e(&var_38)
004034cf        char var_11_1 = 0
004034d4        return std::sys::backtrace::__r...in_short_backtrace::hff9dcd07bd83e4f6()
```

```
int64_t std::sys::backtrace::__rust_begin_short_backtrace::hff9dcd07bd83e4f6()

0040b330    int64_t std::sys::backtrace::__rust_begin_short_backtrace::hff9dcd07bd83e4f6()

0040b33e        return core::ops::function::FnOnce::call_once::hb2a496169abf6687()
```

```
int64_t core::ops::function::FnOnce::call_once::hb2a496169abf6687()

00408f30    int64_t core::ops::function::FnOnce::call_once::hb2a496169abf6687()

00408f3e        return ldr::omg::h298374313dff0ecb()
```

Finally, the actual thread entry point: `ldr::omg`

```
int64_t ldr::omg::h298374313dff0ecb()

00405720    int64_t ldr::omg::h298374313dff0ecb()

00405720        int64_t rbx
00405720        int64_t var_8 = rbx
0040573d        void var_248
0040573d        alloc::slice::<impl [T]>::to_vec::hca02de8bb68fb008(&var_248, data: &data_4f5898,
0040573d            data_len: 0x2df5856)
00405752        char s_1[0x40]
00405752        _.text(s: &s_1, c: 0, n: 0x40)
00405757        int32_t var_2c = 0xe103659f
00405757
00405777        for (int64_t i = 0; i u< 0x40; i += 4) {
00405806            int32_t rax_12 = var_2c ^ var_2c << 0xd
0040581b            int32_t rax_14 = rax_12 ^ rax_12 u>> 0x11
00405840            var_2c = rax_14 ^ rax_14 << 5
0040584e            int32_t rax_18 = core::num::<impl u32>::to_ne_bytes::ha38784835d9da863(var_2c)
00405878            int32_t rax_20
00405878            rax_20.b = rax_18.b
00405878
0040589a            if (i u>= 0x40) {
004058f0                core::panicking::panic_bounds_check::ha307efadc328ce06()
004058f0                noreturn
0040589a            }
0040589a
004058ab            s_1[i] = rax_20.b
004058b2            int64_t rax_22
004058b2            rax_22.b = rax_18:1.b
004058b2
```


hmm the ldr:: functions look interesting
i'll look at ldr::omg
wow great i found the loader code!
std::thread::spawn
std::sys::pal::windows::thread::Thread::new
CreateThread
&dyn std::ops::FnOnce
VTABLES

But try doing it again, without symbols

Malware authors often still forget to strip debug symbols from Rust binaries, because the `release` build profile of the Rust toolchain still includes debug symbols by default.

However, changing this is as simple as putting the following in `Cargo.toml` (the build configuration file)

```
strip = "true"
```

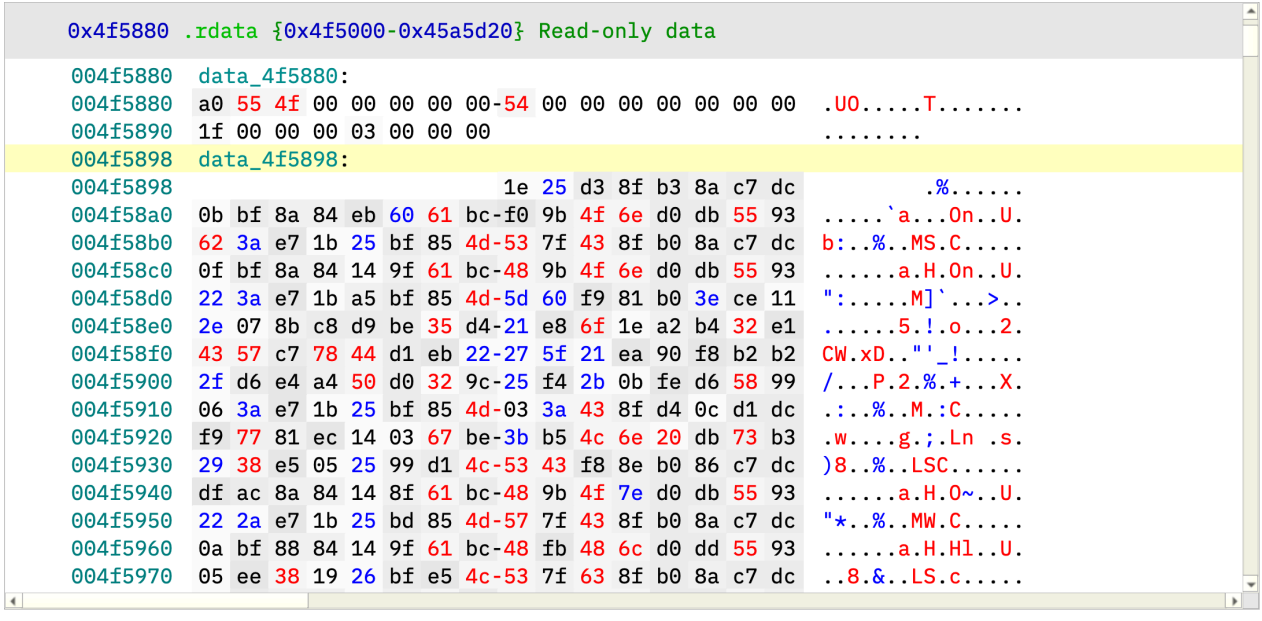
Decrypting the payload

Creating a `Vec` in `ldr::omg`

We already reversed `alloc::slice::<impl [T]>::to_vec` back in `ldr::main`!

```
int64_t ldr::omg::h298374313dff0ecb()

00405720    int64_t ldr::omg::h298374313dff0ecb()

00405720        int64_t rbx
00405720        int64_t var_8 = rbx
0040573d        void var_248
0040573d        alloc::slice::<impl [T]>::to_vec::hca02de8bb68fb008(&var_248,
0040573d            data: &data_4f5898, data_len: 0x2df5856)
00405752        char s_1[0x40]
00405752        _.text(s: &s_1, c: 0, n: 0x40)
00405757        int32_t var_2c = 0xe103659f
00405757
```

A payload at `0x4f5898`, with length `0x2df5856`



```
0x4f5880 .rdata {0x4f5000-0x45a5d20} Read-only data

004f5880  data_4f5880:
004f5880  a0 55 4f 00 00 00 00 00-54 00 00 00 00 00 00 00  .UO.....T.......
004f5890  1f 00 00 00 03 00 00 00                          ........
004f5898  data_4f5898:
004f5898                              1e 25 d3 8f b3 8a c7 dc          .%......
004f58a0  0b bf 8a 84 eb 60 61 bc-f0 9b 4f 6e d0 db 55 93  .....`a...On..U.
004f58b0  62 3a e7 1b 25 bf 85 4d-53 7f 43 8f b0 8a c7 dc  b:..%..MS.C.....
004f58c0  0f bf 8a 84 14 9f 61 bc-48 9b 4f 6e d0 db 55 93  ......a.H.On..U.
004f58d0  22 3a e7 1b a5 bf 85 4d-5d 60 f9 81 b0 3e ce 11  ":.....M]`...>..
004f58e0  2e 07 8b c8 d9 be 35 d4-21 e8 6f 1e a2 b4 32 e1  ......5.!.o...2.
004f58f0  43 57 c7 78 44 d1 eb 22-27 5f 21 ea 90 f8 b2 b2  CW.xD..."'_!.....
004f5900  2f d6 e4 a4 50 d0 32 9c-25 f4 2b 0b fe d6 58 99  /...P.2.%.+...X.
004f5910  06 3a e7 1b 25 bf 85 4d-03 3a 43 8f d4 0c d1 dc  .:..%..M.:C.....
004f5920  f9 77 81 ec 14 03 67 be-3b b5 4c 6e 20 db 73 b3  .w....g.;.Ln .s.
004f5930  29 38 e5 05 25 99 d1 4c-53 43 f8 8e b0 86 c7 dc  )8..%..LSC......
004f5940  df ac 8a 84 14 8f 61 bc-48 9b 4f 7e d0 db 55 93  ......a.H.O~..U.
004f5950  22 2a e7 1b 25 bd 85 4d-57 7f 43 8f b0 8a c7 dc  "*..%..MW.C.....
004f5960  0a bf 88 84 14 9f 61 bc-48 fb 48 6c d0 dd 55 93  ......a.H.Hl..U.
004f5970  05 ee 38 19 26 bf e5 4c-53 7f 63 8f b0 8a c7 dc  ..8.&..LS.c.....
```

Carving out the payload at `0x4f5898`, with length `0x2df5856`

```
93161dfbb081e88b2b0cb2ba294c7c7baaf049ba455aaf0b15df7e08539e2109
```

Fairly high entropy (97%), but the entropy is not uniform:

Creating a `Vec` in `ldr::omg`

```
int64_t ldr::omg::h298374313dff0ecb()

00405720    int64_t ldr::omg::h298374313dff0ecb()

00405720        int64_t rbx
00405720        int64_t var_8 = rbx
0040573d        struct std::vec::Vec encrypted_payload_vec
0040573d        alloc::slice::<impl [T]>::to_vec::hca02de8bb68fb008(
0040573d            &encrypted_payload_vec, data: &data_4f5898, data_len: 0x2df5856)
00405752        char s_1[0x40]
00405752        _.text(s: &s_1, c: 0, n: 0x40)
00405757        int32_t var_2c = 0xe103659f
00405757
00405777        for (int64_t i = 0; i u< 0x40; i += 4) {
00405806            int32_t rax_12 = var_2c ^ var_2c << 0xd
0040581b            int32_t rax_14 = rax_12 ^ rax_12 u>> 0x11
00405840            var_2c = rax_14 ^ rax_14 << 5
0040584e            int32_t rax_18 =
0040584e                core::num::<impl u32>::to_ne_bytes::ha38784835d9da863(var_2c)
00405878            int32_t rax_20
```

Some interesting symbols in `ldr::omg`

```
int64_t ldr::omg::h298374313dff0ecb()

00405a6b    }
00405a6b
004061b7    void var_168
004061b7    memcpy(dest: &var_168, src: &s, n: 0x40)
004061d2    void var_1f8
004061d2    memcpy(dest: &var_1f8, src: &var_168, n: 0x40)
00406230    int64_t var_218[0x4]
00406230    include_crypt_crypto::key::EncryptionKey::new::hb945f29bab485494(&var_218,
00406230        obfstr::ObfBuffer<[u8; LEN]>::as_str::hab897014783120cc(&var_1f8), 0x40, 0x20)
0040624f    int64_t var_230[0x3]
0040624f    core::result::Result<T,E>::map::h0d17ea1db1da02a3(&var_230, &var_218,
004062e3    int16_t* rax_106
004062e3    int64_t rdx_23
004062e3    rax_106, rdx_23 =
004062e3        <alloc::vec::Vec<T,A> as...ref::Deref>::deref::hb7df086d509addab(&var_298)
0040635a    int64_t var_280 = mexec::mexec::h0a728fff25aaa356(rax_106, rdx_23, 0, 1, 0)
00406378    return core::ptr::drop_in_place<alloc::vec::Vec<u8>>::hd93f860cc4a31edd(&var_298)
```

Figuring out library version information via panic metadata

```
int64_t ldr::omg::h298374313dff0ecb()

00405878
0040589a        if (i u>= 0x40) {
004058f0            core::panicking::panic_bounds_check::ha307efadc328ce06()
004058f0            noreturn
0040589a        }
0040589a
004058ab        s_1[i] = rax_20.b
004058b2        int64_t rax_22
004058b2        rax_22.b = rax_18:1.b
004058b2
004058d8        if (i + 1 u>= 0x40) {
0040594d            core::panicking::panic_bounds_check::ha307efadc328ce06()
0040594d            noreturn
004058d8        }
004058d8
```

Figuring out library information via panic metadata

```
int64_t ldr::omg::h298374313dff0ecb()

004058d8  721f              jb      0x4058f9

004058da  eb5d              jmp     0x405939

004058dc  488b8c2490010000  mov     rcx, qword [rsp+0x190 {var_2a8_1}]
004058e4  4c8d054dff0e00    lea     r8, [rel data_4f5838]
004058eb  ba40000000        mov     edx, 0x40
004058f0  e8b3620d00        call    core::panicking::panic_bounds_check::ha307efadc328ce06
{ Does not return }
```

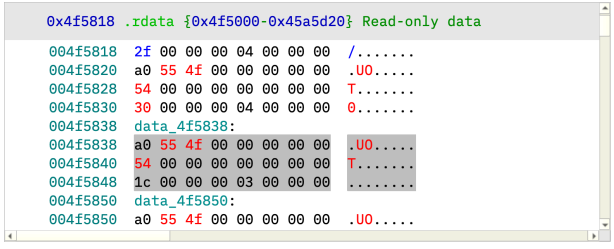
Figuring out library information via panic metadata

cxiao.net - Using panic metadata to recover source code information from Rust binaries (2023-12-08)

```
0x4f5818 .rdata {0x4f5000-0x45a5d20} Read-only data

004f5818  2f 00 00 00 04 00 00 00  /.......
004f5820  a0 55 4f 00 00 00 00 00  .UO.....
004f5828  54 00 00 00 00 00 00 00  T.......
004f5830  30 00 00 00 04 00 00 00  0.......
004f5838  data_4f5838:
004f5838  a0 55 4f 00 00 00 00 00  .UO.....
004f5840  54 00 00 00 00 00 00 00  T.......
004f5848  1c 00 00 00 03 00 00 00  ........
004f5850  data_4f5850:
004f5850  a0 55 4f 00 00 00 00 00  .UO.....
```

Figuring out library information via panic metadata

```
struct `core::panic::Location`
{
    struct `&str`
    {
        char* ptr;
        uint64_t len;
    } file;
    uint32_t line;
    uint32_t col;
};
```



Figuring out library information via panic metadata

The metadata tells us:

- Library: `obfstr`

> Compiletime string constant obfuscation for Rust.

- Version: `0.2.6`
- Source file: `src/bytes.rs`
- Line number: 28 (`0x1c`)
- Column number: 3

Source code: obfstr/0.2.6/source/src/bytes.rs#28

- This is inside `fn keystream` :

```
/// Generate the key stream for array of given length.
#[inline(always)]
fn keystream<const LEN: usize>(key: u32)
    -> [u8; LEN]
```

```
0x4f5808 .rdata {0x4f5000-0x45a5d20} Read-only data

004f5808  a0 55 4f 00 00 00 00 00  .UO.....
004f5810  54 00 00 00 00 00 00 00  T.......
004f5818  2f 00 00 00 04 00 00 00  /.......
004f5820  a0 55 4f 00 00 00 00 00  .UO.....
004f5828  54 00 00 00 00 00 00 00  T.......
004f5830  30 00 00 00 04 00 00 00  0.......

004f5838  struct core::panic::Location p_obfstr_panic_str =
004f5838  {
004f5838      struct &str file =
004f5838      {
004f5838          char* ptr = obfstr_panic_str {"/root/.cargo/registry/src/inde
004f5840          uint64_t len = 0x54
004f5848      }
004f5848      uint32_t line = 0x1c
004f584c      uint32_t col = 0x3
004f5850  }
```

`obfstr::ObfBuffer` in `obfstr` version 0.2.6

Source code: obfstr/0.2.6/source/src/lib.rs#135

```
/// Deobfuscated string buffer.
#[repr(transparent)]
pub struct ObfBuffer<A: ?Sized>(pub A);
```

> **Note**
>
> This structure doesn't exist in new versions of `obfstr`! Metadata about library versions is useful, even if we have symbols.

```
int64_t ldr::omg::h298374313dff0ecb()

004061b7    void var_168
004061b7    memcpy(dest: &var_168, src: &s, n: 0x40)
004061d2    void var_1f8
004061d2    memcpy(dest: &var_1f8, src: &var_168, n: 0x40)
00406230    int64_t var_218[0x4]
00406230    include_crypt_crypto::key::EncryptionKey::new::hb945f29bab485494(&var_218,
00406230        obfstr::ObfBuffer<[u8; LEN]>::as_str::hab897014783120cc(&var_1f8), 0x40, 0x20)
0040624f    int64_t var_230[0x3]
0040624f    core::result::Result<T,E>::map::h0d17ea1db1da02a3(&var_230, &var_218,
0040624f        &encrypted_payload_vec)
```

include_crypt

`include_crypt` allows a developer to embed encrypted files into their binary.

```rust
use include_crypt::{include_crypt, EncryptedFile};

static FILE: EncryptedFile = include_crypt!("assets/file.txt");

fn main() {
    let decrypted = FILE.decrypt();
    let decrypted_str = FILE.decrypt_str();
}
```

Developers can specify the encryption method, and a custom key:

```rust
// Encrypt using XOR with custom key
let file: EncryptedFile = include_crypt!(XOR, "src/lib.rs", 0xdeadbeef);

// Encrypt using AES with custom key
let file: EncryptedFile = include_crypt!(AES, "src/lib.rs", 0xdeadbeef);
```

include_crypt_crypto::key::EncryptionKey::new

Panic metadata tells us:

- Library: `include_crypt_crypto`
  - This is a dependency of `include_crypt`
- Version: `0.1.0`

Source code: include-crypt-crypto/0.1.0/src/include_crypt_crypto/key.rs.html#14

```rust
/// Creates a new key from the specified hex string.
pub fn new(key: &'_ str, key_len: usize) -> Result<Self, String> {
    // Remove the optional trailing '0x' and convert to vector
    let mut key = hex::decode(key.trim_start_matches("0x"))
        .map_err(|e| e.to_string())?;

    // Extend the key if it is smaller than the default key length.
    if key.len() != key_len {
        key = key.into_iter().cycle().take(DEFAULT_KEY_LEN)
            .collect::<Vec<_>>();
    }

    Ok(Self { data: key })
}
```

The logic of `include_crypt_crypto`, `obfstr`

There are 2 steps happening here:

1. The code generated by `obfstr` compile-time deobfuscates a string.

```
int64_t ldr::omg::h298374313dff0ecb()

00405752    char obfstr_obfuscated_string_[0x40]
00405752    memset(s: &obfstr_obfuscated_string_, c: 0, n: 0x40)
00405757    int32_t var_2c = 0xe103659f
00405757
00405777    // keystream generation for obfstr
00405777    for (int64_t i = 0; i u< 0x40; i += 4) {
00405806        int32_t rax_12 = var_2c ^ var_2c << 0xd
0040581b        int32_t rax_14 = rax_12 ^ rax_12 u>> 0x11
00405840        var_2c = rax_14 ^ rax_14 << 5
0040584e        int32_t rax_18 = core::num::<impl u32>::to_ne_bytes::ha38784835d9da863(var_2c)
00405878        int32_t rax_20
00405878        rax_20.b = rax_18.b
00405878
0040589a        // panic inside keystream generation for obfstr
0040589a        if (i u>= 0x40) {
004058f0            core::panicking::panic_bounds_check::ha307efadc328ce06()
004058f0            noreturn
0040589a        }
0040589a
004058ab        obfstr_obfuscated_string_[i] = rax_20.b
```

The logic of `include_crypt_crypto`, `obfstr`

2. The deobfuscated string is hex-decoded by `include_crypt_crypto::key::EncryptionKey::new`, into bytes. This is then used as the decryption key for the payload.

```
int64_t ldr::omg::h298374313dff0ecb()

00405a6b
004061b7    char tmp[0x40]
004061b7    memcpy(dest: &tmp, src: &obfstr_deobfuscated_string, n: 0x40)
004061d2    char obfstr_deobfuscated_string_[0x40]
004061d2    memcpy(dest: &obfstr_deobfuscated_string_, src: &tmp, n: 0x40)
00406230    int64_t encryption_key_bytes[0x4]
00406230    include_crypt_crypto::key::EncryptionKey::new::hb945f29bab485494(
00406230        &encryption_key_bytes,
00406230        obfstr::ObfBuffer<[u8; LEN]>::as_str::hab897014783120cc(
00406230            &obfstr_deobfuscated_string_),
00406230        0x40, 0x20)
0040624f    int64_t var_230[0x3]
```

Getting the `obfstr` -deobfuscated string

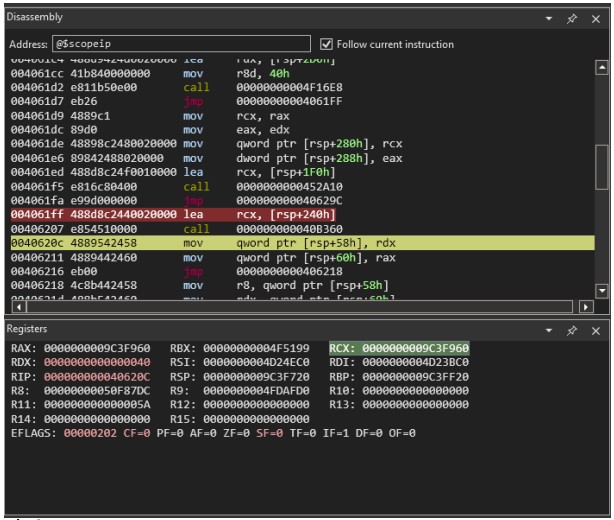
We can statically calculate, emulate, or just debug the binary (which is what I did):

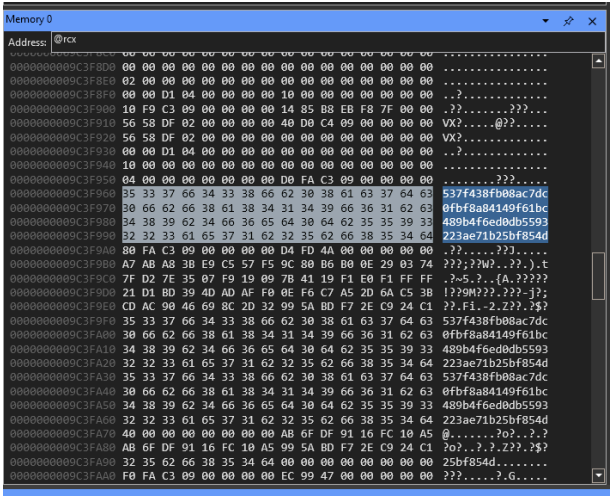
- I set a breakpoint on virtual address `0x4061ff`
- This is the instruction that passes the deobfuscated string into `obfstr::ObfBuffer<[u8; LEN]>::as_str` , via register `rcx` :

```
int64_t ldr::omg::h298374313dff0ecb()

004061a1  488d8c24d0020000   lea     rcx, [rsp+0x2d0 {tmp}]
004061a9  488d942410030000   lea     rdx, [rsp+0x310 {obfstr_deobfuscated_string}]
004061b1  41b840000000       mov     r8d, 0x40
004061b7  e82cb50e00         call    memcpy
004061bc  488d8c2440020000   lea     rcx, [rsp+0x240 {obfstr_deobfuscated_string_}]
004061c4  488d9424d0020000   lea     rdx, [rsp+0x2d0 {tmp}]
004061cc  41b840000000       mov     r8d, 0x40
004061d2  e811b50e00         call    memcpy
004061d7  eb26               jmp     0x4061ff

004061ff  488d8c2440020000   lea     rcx, [rsp+0x240 {obfstr_deobfuscated_string_}]
00406207  e854510000         call    obfstr::ObfBuffer<[u8; LEN]>::as_str::hab897014783120cc
0040620c  4889542458         mov     qword [rsp+0x58 {var_3e0}], rdx  {0x40}
00406211  4889442460         mov     qword [rsp+0x60 {var_3d8}], rax
00406216  eb00               jmp     0x406218
```

Getting the `obfstr` -deobfuscated string, via debugging

At virtual address `0x4061ff`, we can examine the value at the memory location in `rcx` for the decrypted string.





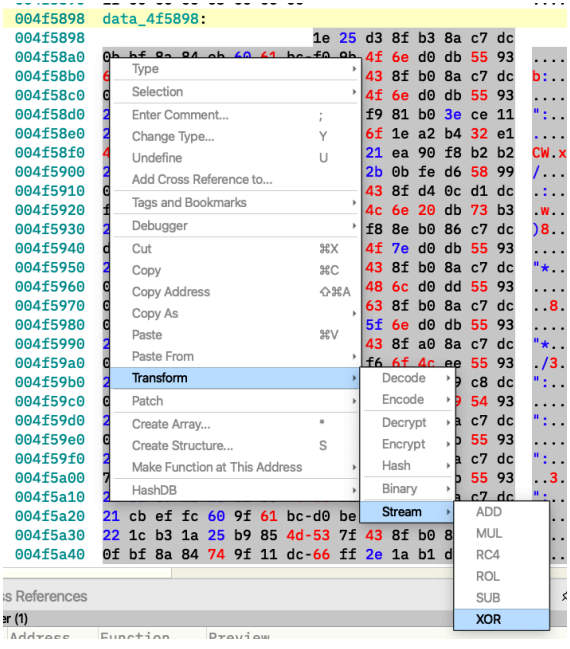
Decrypting the payload

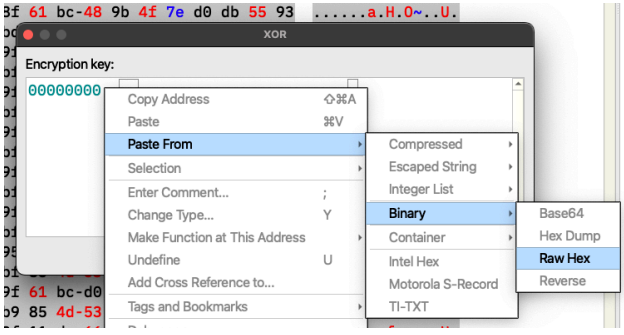
The deobfuscated string is

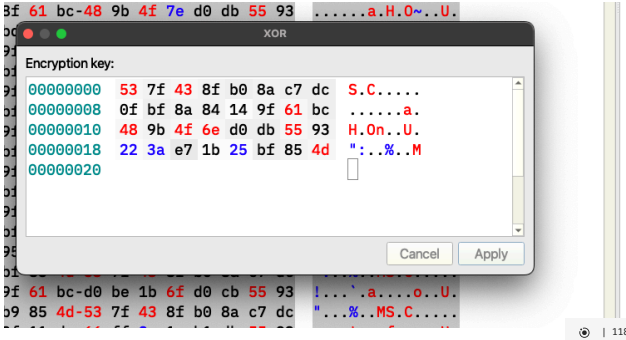
```
537f438fb08ac7dc0fbf8a84149f61bc489b4f6ed0db5593223ae71b25bf854d
```

This, once hex-decoded, is our XOR key for the payload!

Decrypting the payload







Finally, the payload

```
0x4f5860 .rdata {0x4f5000-0x45a5d20} Read-only data

004f5860  1d 00 00 00 03 00 00 00                          ........
004f5868  data_4f5868:
004f5868                           a0 55 4f 00 00 00 00 00          .UO.....
004f5870  54 00 00 00 00 00 00 00-1e 00 00 00 03 00 00 00  T...............
004f5880  data_4f5880:
004f5880  a0 55 4f 00 00 00 00 00-54 00 00 00 00 00 00 00  .UO.....T.......
004f5890  1f 00 00 00 03 00 00 00                          ........
004f5898  data_4f5898:
004f5898                           4d 5a 90 00 03 00 00 00          MZ......
004f58a0  04 00 00 00 ff ff 00 00-b8 00 00 00 00 00 00 00  ................
004f58b0  40 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  @...............
004f58c0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
004f58d0  00 00 00 00 80 00 00 00-0e 1f ba 0e 00 b4 09 cd  ................
004f58e0  21 b8 01 4c cd 21 54 68-69 73 20 70 72 6f 67 72  !..L.!This progr
004f58f0  61 6d 20 63 61 6e 6e 6f-74 20 62 65 20 72 75 6e  am cannot be run
004f5900  20 69 6e 20 44 4f 53 20-6d 6f 64 65 2e 0d 0d 0a   in DOS mode....
004f5910  24 00 00 00 00 00 00 00-50 45 00 00 64 86 16 00  $.......PE..d...
004f5920  f6 c8 0b 68 00 9c 06 02-73 2e 03 00 f0 00 26 20  ...h....s.....&
004f5930  0b 02 02 1e 00 26 54 01-00 3c bb 01 00 0c 00 00  .....&T..<......
004f5940  d0 13 00 00 00 10 00 00-00 00 00 10 00 00 00 00  ................
004f5950  00 10 00 00 00 02 00 00-04 00 00 00 00 00 00 00  ................
004f5960  05 00 02 00 00 00 00 00-00 60 07 02 00 06 00 00  .........`......
004f5970  27 d4 df 02 03 00 60 01-00 00 20 00 00 00 00 00  '.....`... .....
004f5980  00 10 00 00 00 00 00 00-00 00 00 10 00 00 00 00  ................
004f5990  00 10 00 00 00 00 00 00-00 00 00 00 10 00 00 00  ................
004f59a0  00 90 b9 01 5f 00 00 00-00 a0 b9 01 9c 35 00 00  ...._.........5..
004f59b0  00 00 00 00 00 00 00 00-00 60 95 01 b4 63 0f 00  .........`...c..
004f59c0  00 00 00 00 00 00 00 00-00 00 ba 01 14 a2 01 00  ................
004f59d0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
004f59e0  00 00 00 00 00 00 00 00-20 f0 b9 01 28 00 00 00            (...
```

What is the payload?

Windows x86_64 DLL written in Rust (MalwareBazaar download)

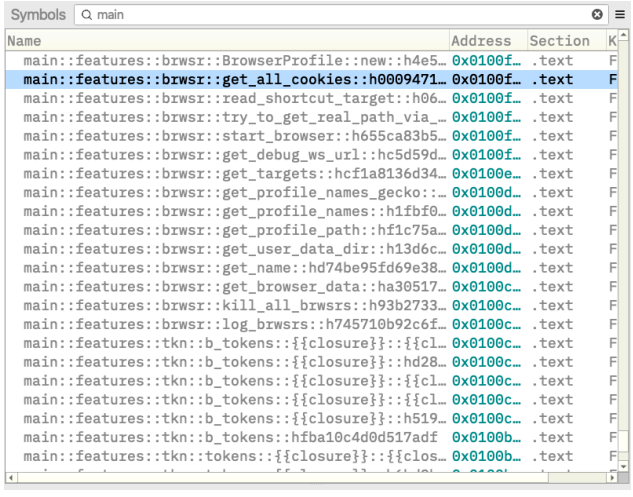
```
66054607f38481ee7e39e002b58fe950966c4c0203df39f46acfe5c0e857c89a
```

All debug symbols are intact.

This is an infostealer called *Myth Stealer*. It mainly targets browser data for exfiltration, including:

- Cookies
- Autofill data
- Stored password data
- History

> **Note**
>
> For more information about *Myth Stealer*, check out the 2025-06-05 article from Trellix, *Demystifying Myth Stealer: A Rust Based InfoStealer*!



Conclusion

- Even a very simple Rust binary with full debug symbols can be very complex to reverse.
- The Rust standard library contains A LOT of code, even in very common standard library functions
- The Rust third-party library ecosystem is quite mature, with utility libraries available for most common malware functionality.

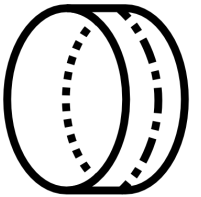
And finally, most importantly:

We need to understand the Rust language to have any hope of reversing Rust binaries!

Special Thanks

- Thank you to @0xabad1dea@infosec.exchange and @demize@unstable.systems for providing the original sample.
- Thank you to one of the authors of the 2025-06-05 article from Trellix, *Demystifying Myth Stealer: A Rust Based InfoStealer*, for reaching out and informing me about Myth Stealer.
  - (They chose to remain anonymous, but they know who they are 👊)
- Illustrations used in the slides are from 19th century French artist Eugène Flandin.

Interested in training?



Decoder Loop

contact@decoderloop.com