Recovering Inlined Function

For Rust Reversing

Who are we?

- **Bryton Bernard** (@lxt33r)
  - Security Engineer
  - Junior Reverse Engineer
  - CTF player for MadeinFrance & TheHackersCrew (lextersec.fr)
- Specialized in
  - Reverse engineering, De-Obfuscation
  - Rust, Go,
  - Binary exploitation

- **Mathieu Hoste (**@mhoste1**)**
  - Security Engineer
  - Junior Reverse Engineer
  - CTF player for MadeinFrance & TheHackersCrew
- Specialized in
  - Reverse engineering,
  - Rust,
  - Binary exploitation

Who is Fuzzinglabs?

Trainings

- **Rust** Security Audit & Fuzzing
- **Go** Security Audit & Fuzzing
- **WebAssembly** Reversing
- **C/C++** Whitebox Fuzzing
- Practical Web **Browser** Fuzzing

Services/Products

- Audit & **Fuzzing**
- Security Engineering
  - Open-source tools
  - Closed-source products
- Domains
  - Blockchain
  - OSINT
  - Browser
  - Telecommunication
  - Hardware

Research

- Youtube
  - ~6k subscribers
  - 60+ videos
- **Public talks**/Trainings
  - BlackHat USA, REcon, OffensiveCon, RingZer0, PoC, ToorCon, hack.lu, NorthSec, SSTIC, etc.

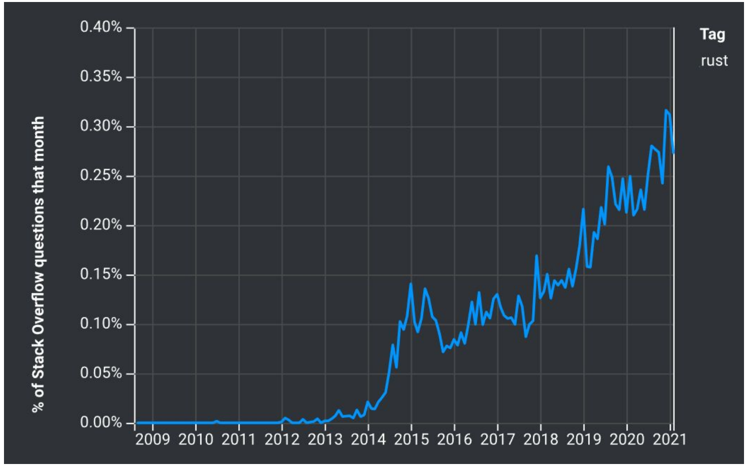
Summary

1. Little Presentation on Rust
2. Discussing Rust Reverse Engineering
3. Specificities and Challenges of Rust Reverse Engineering
4. Inlining
5. Solutions for recovering inlined functions
6. PoC for automation
7. Conclusion and future

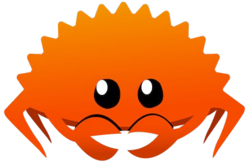
What’s Rust?

- Rust is a systems **programming language** created by Mozilla
  - Syntactically similar to C++
  - Safer alternative to C and C++
  - Focused on **performance** and safety, especially **safe concurrency**.
  - Provides **memory safety** without **using garbage collection.**

```rust
fn main() {
    println!("Hello, World!");
}
```



- Growing community (aka Rustaceans)
  - **Most loved language** for 8 year now (StackOverflow survey)



Who is using Rust for CyberSecurity?

- **Windows Kernel** in rust
  - “30 year old code killed! Microsoft rewrites Windows kernel with 180,000 lines of Rust.”
  - “Microsoft announced that the latest Windows 11 […] memory safety-focused Rust programming language.”

- Adoption by **malware authors**
  - Static linking
  - Support for many os
  - Minimal dependencies

- **Challenge** for reverse engineer

- Features for **malware development**
  - **Memory safety** : lot of ransom group like lazarus.
  - **Zero-Cost Abstractions**: Rust's efficient abstractions allow malware to be both fast and compact, evading detection.

Rust Reversing 101

Best (free) Disassemblers for Rust

- **BINARY NINJA**
  - ✅ Community
  - ✅ Good API
  - ✅ Intermediate representation
  - ❌ Bad at strings analysis
- **IDA FREE**
  - ✅ Optimized decompiler for Rust
  - ✅ Support big binary size
  - ❌ Bad at strings analysis
- **Ghidra**
  - ✅ New features for Rust
  - ✅ Community
  - ❌ Doesn’t support big binary size
  - ❌ Calling convention
  - ❌ Java API, Undocumented python binding

Rust Reversing 101: Symbols

- **Symbols**
    - **Mangling:** by the compiler to generate **unique names** for functions.
    - **Demangling**: Tools like `rustc-demangle` convert mangled names back to human-readable forms for debugging.

```
core :: str :: pattern :: simd_contains :: _$u7b$$u7b$closure$u7d$$u7d$ :: h5685b91fbce861cf
Crate   Module  Submodule   Method          Closure                           Unique identifier
```

C

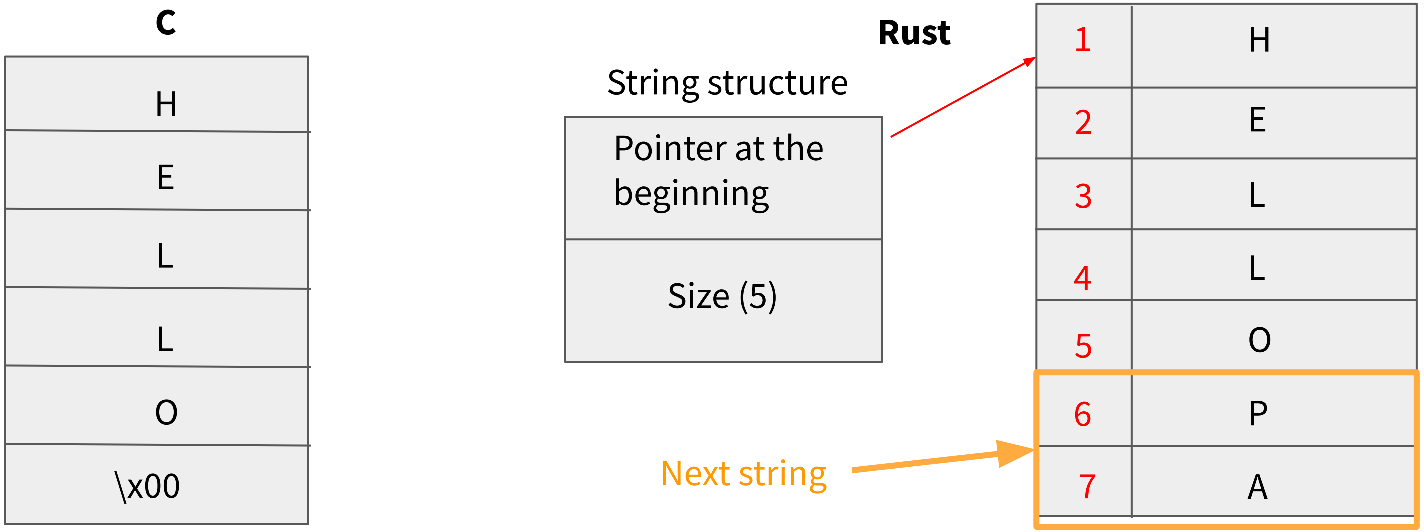
```
_init
sub_1020
sub_1030
__cxa_finalize
puts
_start
deregister_tm_clones
register_tm_clones
__do_global_dtors_aux
frame_dummy
main
```

Rust

```
_init
sub_6020
sub_6036
_Unwind_Resume
sub_6046
__cxa_finalize
core::str::pattern::simd_contains::_$u7b$$u7b$closure$u7d$$u7d$::h5685b91fbce861cf
core::slice::sort::break_patterns::h3a1e4a7f941aca89
core::slice::sort::partial_insertion_sort::h93333094f90d6425
core::slice::sort::heapsort::h0d4318d5554b67c1
core::panicking::assert_failed::h48c8b6d62e47947b
```

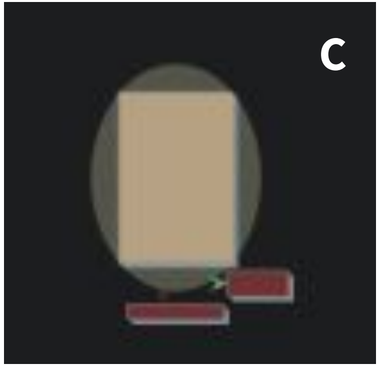
Rust Reversing 101: Strings

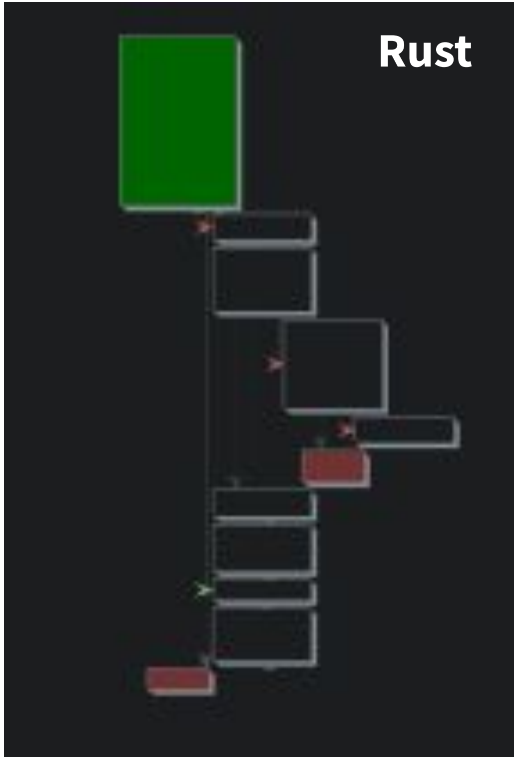
- Rust Strings delimitation are differents
  - Non null terminated byte
  - In **Rust** each **strings** have a **structure** with the **begin pointer** and the corresponding **size** string



Rust Reversing 101: Control Flow

- **Control flow graph**
  - Rust is more verbose
  - Inlining can add a lot of basic Block for the CFG
- **Memory Safety and Abstractions**
  - high-level abstractions : **additional code** at compilation
- **Error Handling**
  - Result, panic, option etc
  - Error paths are explicitly represented in the code
  - Generate additional code to handle different error cases





Rust Reversing 101: Rust Assembly

- **Safety Features**
  - Complex **ownership** and **type systems**
- **Abstractions**
  - Higher-level abstractions, **code concise** but adding complexity
- **Compiler Optimizations**
  - **Optimized machine code**
- **Monomorphization**
  - **Creating specialized** versions of a **generic function** for each **specific type** with which it is used in the source code.
- **Ownership and Borrowing**
  - **Ownership** and **borrowing** model affects **memory management**

Rust Reversing 101: Rust Assembly

- **Safety Features**

```rust
fn id<T>(x: T) -> T {
    return x;
}

fn main() {
    let int = id(10);
    let string = id("some text");
    println!("{int}, {string}");
}
```

mplexity

ce code.

- **Ownership** and **borrowing** model affects **memory management**

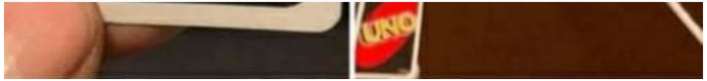
FUZZING LABS

Rust Reversing 101: Rust Assembly

```rust
fn id<T>(x: T) -> T {
    return x;
}

fn main() {
    let int = id(10);
    let string = id("some text");
    println!("{int}, {string}");
}
```

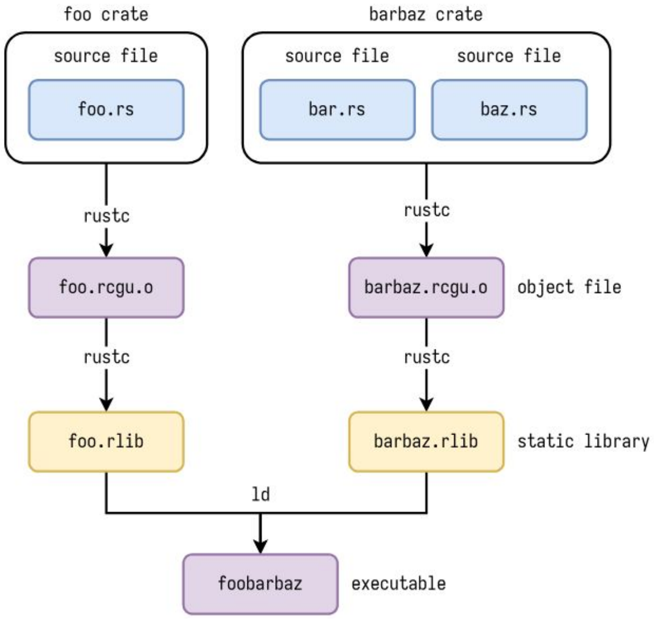
```rust
fn id_i32(x: i32) -> i32 {
    return x;
}

fn id_str(x: &str) -> &str {
    return x;
}

fn main() {
    let int = id_i32(10);
    let string = id_str("some text");
    println!("{int}, {string}");
}
```

- **Ownership and Borrowing**
    - **Ownership** and **borrowing** model affects **memory m**



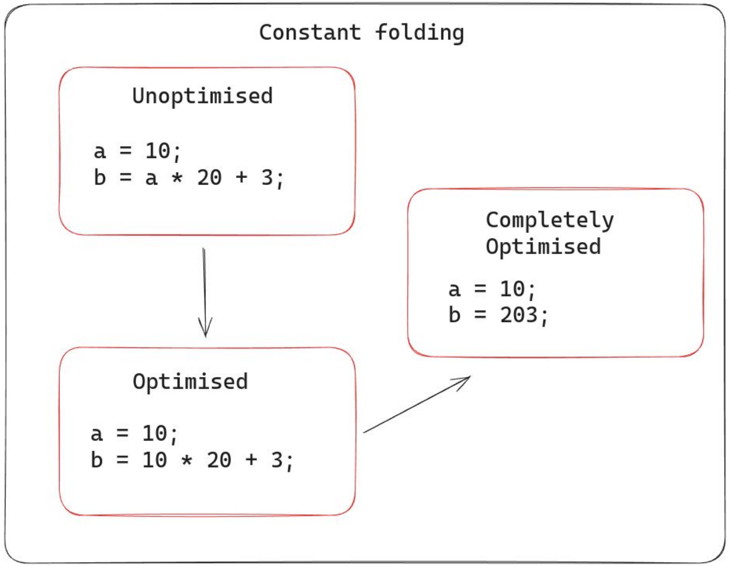
Rust Reversing 101: Static Linking

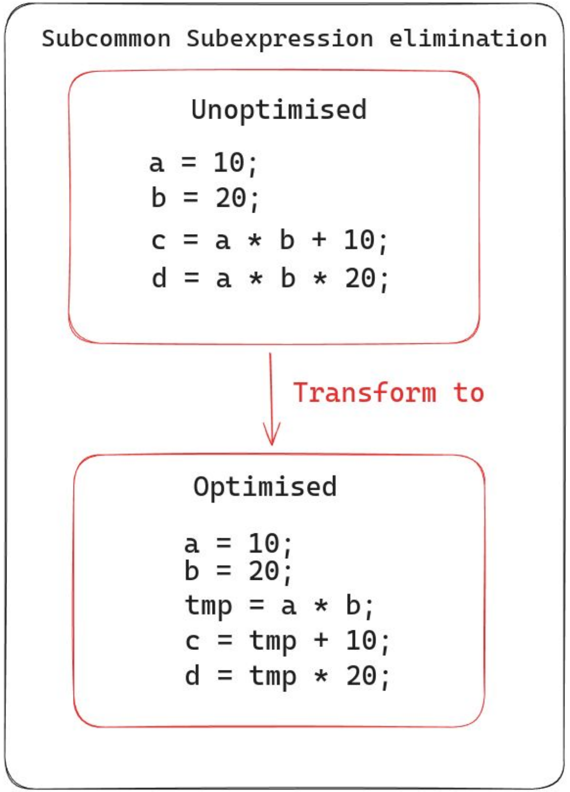
- **Embeds** all necessary **libraries** directly into the executable at compile time
  - More challenging by **eliminating external dependencies**
- All the **necessary code** is contained within a **single executable**
  - **Reducing the number of files needed**
- Compile a Rust program for static linking
  - For default glibc
    - rustc -C target-feature=+crt-static foo.rs



Rust Reversing 101: Optimizations

- **rustc** offers many **features** to **optimize** the **assembly code**
  - **Constant Folding**
  - **Subcommon Subexpression elimination**
  - **Inlining**





Rust Inlining

What is inlining?

Caller of inlined function

```
int a(int y) {
    return b(y) + b(0) + b(y+1);
}
```

Function to inline

```
int b(int x) {
    if (x == 0)
        return 0;
    else
        return x - 1;
}
```

What is inlining?

Caller of inlined function

```
int a(int y) {
    return b(y) + b(0) + b(y+1);
}
```

Function to inline

```
int b(int x) {
    if (x == 0)
        return 0;
    else
        return x - 1;
}
```

Unoptimized inlined function

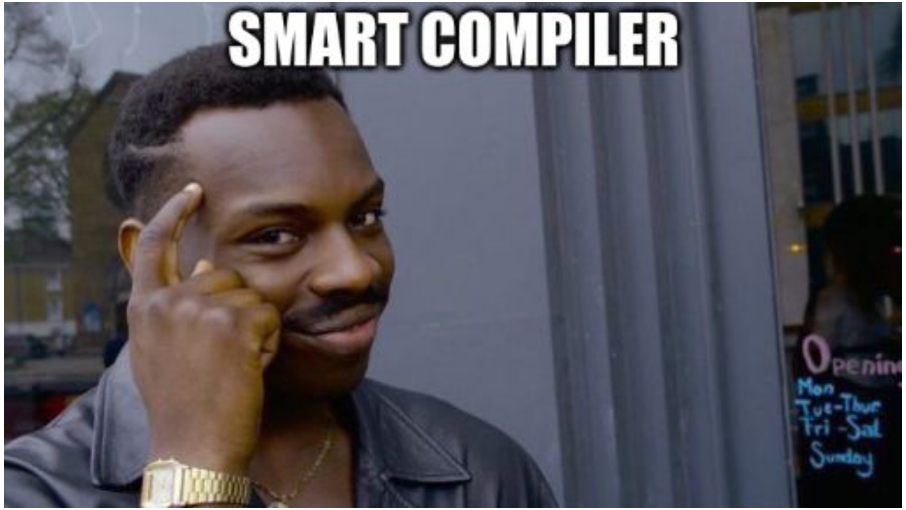
```
int a(int y) {
    int temp = 0;
    if (y   == 0) temp += 0; else temp += y       - 1;
    if (0   == 0) temp += 0; else temp += 0       - 1;
    if (y+1 == 0) temp += 0; else temp += (y + 1) - 1;
    return temp;
}
```

Optimized inlined function

```
int a(int y) {
    if (y == 0)
        return y;
    else if (y == -1)
        return y - 1;
    else
        return y + y - 1;
}
```

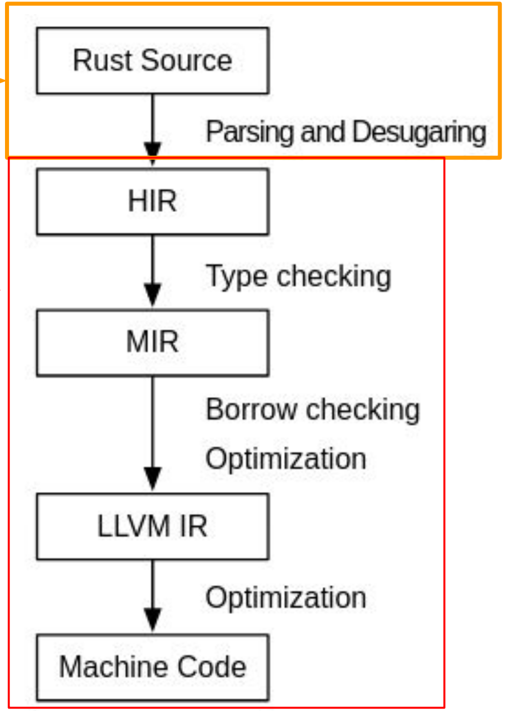
Usage of Inlining

- **Reduced overhead**: Eliminates function call runtime costs.
- **Better optimization**: Allows for more aggressive code optimization.
- **Faster execution**: Speeds up critical paths by removing call/return sequences.
- **Smaller code size:** In some cases, can lead to smaller executable sizes.



Rust Compilation processus

- The process of **compilation** can be splitted in two phases:
    - **Parsing**
    - **Machine code generation**

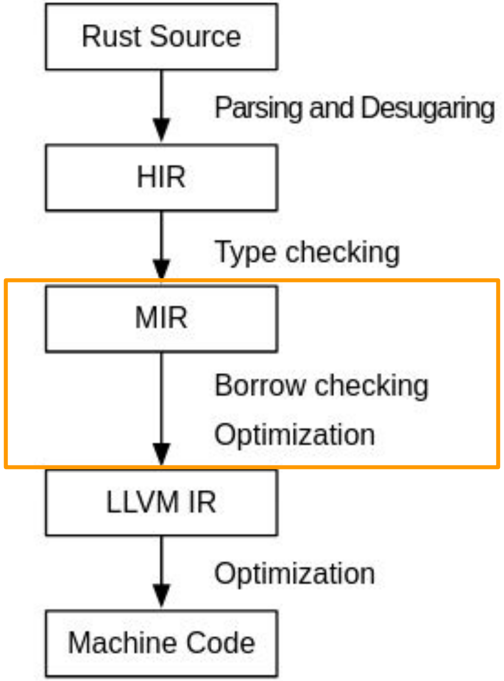


Rust Compilation processus - MIR

- **MIR** : Mid-Level Intermediate Representation.
  - **Borrow checking is done on this representation.**
  - Viewing MIR :
    - `rustc main.rs --emit=mir`

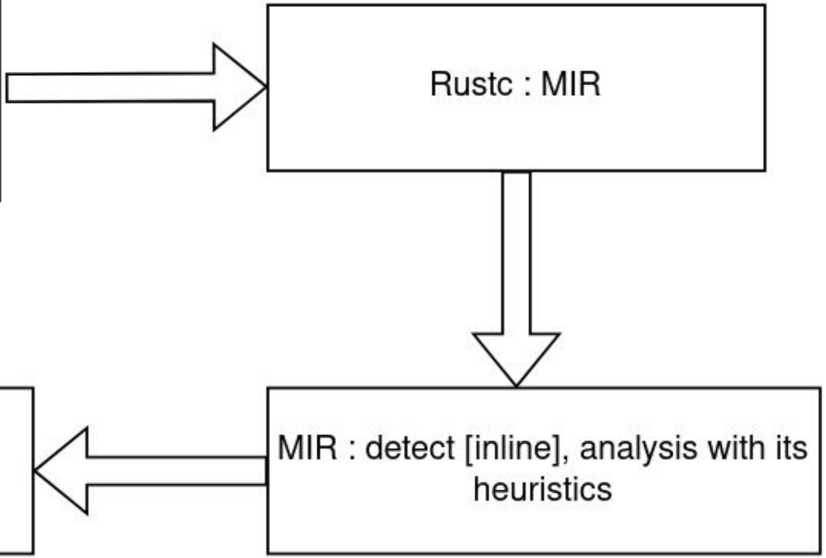
- This part handle inlining decision.

```
#[inline]
#[must_use]
#[stable(feature = "rust1", since = "1.0.0")]
pub unsafe fn from_utf8_unchecked(bytes: Vec<u8>) -> String {
    String { vec: bytes }
}
```



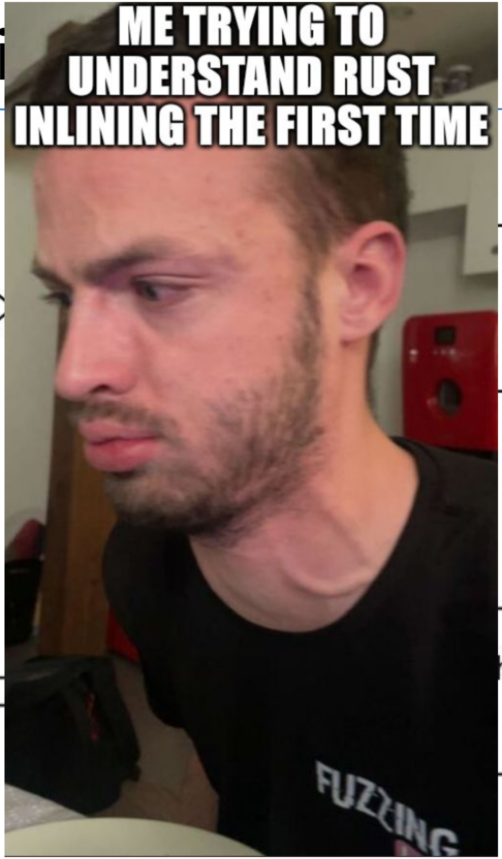
Rust Compilation processus - MIR inlining phase

```rust
#[inline]
#[must_use]
#[stable(feature = "rust1", since = "1.0.0")]
pub unsafe fn from_utf8_unchecked(bytes: Vec<u8>) -> String {
    String { vec: bytes }
}
```



Rust Compilation processus - MIR inlini

```rust
#[inline]
#[must_use]
#[stable(feature = "rust1", since = "1.0.0")]
pub unsafe fn from_utf8_unchecked(bytes: Vec<u8>) -> String {
    String { vec: bytes }
}
```



h its

Gives LLVM pass

LLVM : Perform inlining

EXAMPLE: Simple UTF-8 Checker

- Ask for an **input** and **store** it as a **String**

- Call **expect()**
  - to check the **Read_line()** success

- Call **trim()**
  - to remove bad chars like ‘**\n**’, ‘**\t**’ etc.

- Call **from_utf8()**
  - to check the **UTF8 validity** of our input

```rust
use std::io;
use std::str;

fn my_function(input: &[u8]) -> Result<&str, str::Utf8Error> {
    str::from_utf8(input)
}

fn main() {
    println!("Enter an input :");

    let mut input = String::new();
    io::stdin().read_line(&mut input).expect("Error reading the input");

    let bytes = input.trim().as_bytes();

    match my_function(bytes) {
        Ok(s) => println!("UTF-8 valide : {}", s),
        Err(e) => println!("UTF-8 invalide, looser : {:?}", e)
    }
}
```

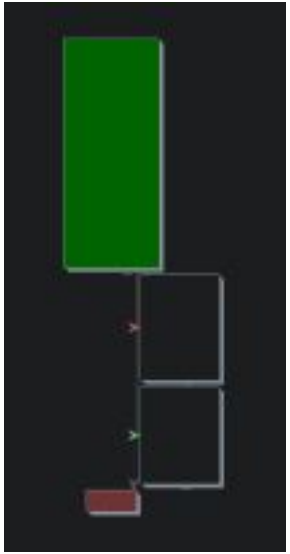
EXAMPLE: CFG

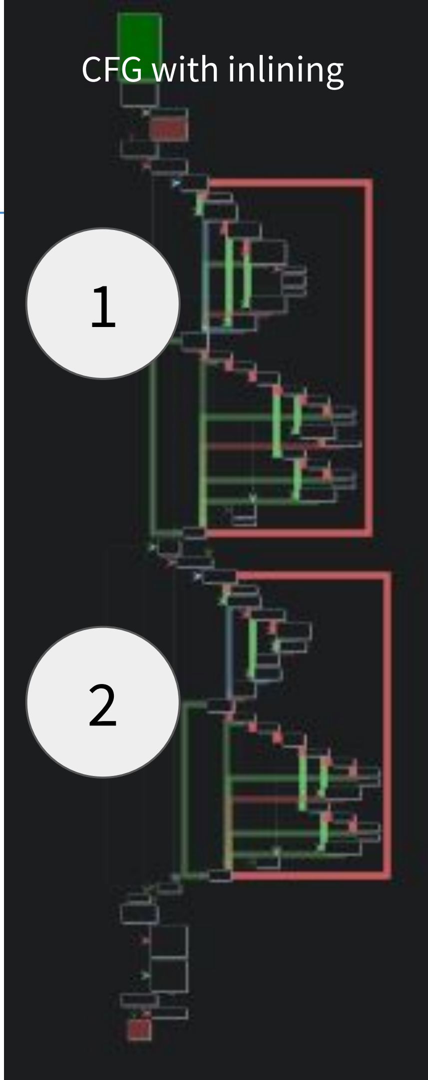
- Two control flow graphs :
    - **With inlining** : release mode
        - Opt-level = 3
        - Debug = false
        - LTO = activate ([Link Time Optimization](#))
        - Panic = unwind. etc.
    - **Without inlining** : debug mode
        - opt-level = 0
        - debug = true
        - lto = false
        - panic = unwind. etc.

- A lot of code has been added
    - More verbose
    - Bigger CFG

- **Block 1** and **block 2** are possibly inlined functions

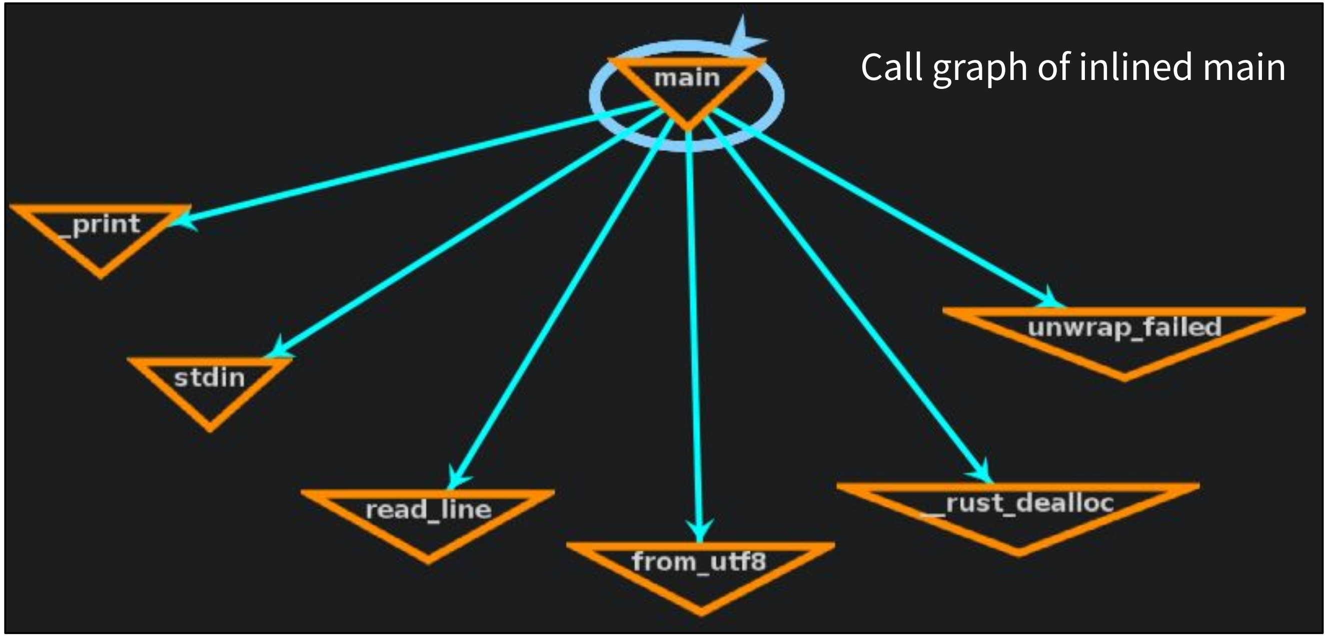
Basic CFG





EXAMPLE: Call graph

- Some **calls** are **missing** compared to **initial source code**
  - **my_function**()
  - **trim**()
  - **expect**()
  - **as_bytes**()



Inlining difficulties

- Calls are missing but the **code is still here**
- We see the **source code** of each functions
- Code very hard to read and understand
  - Complicated operations
  - Difficult Code flow
  - **Not useful to try to understand** the code because It is a Rust native function
- Take a lot of time

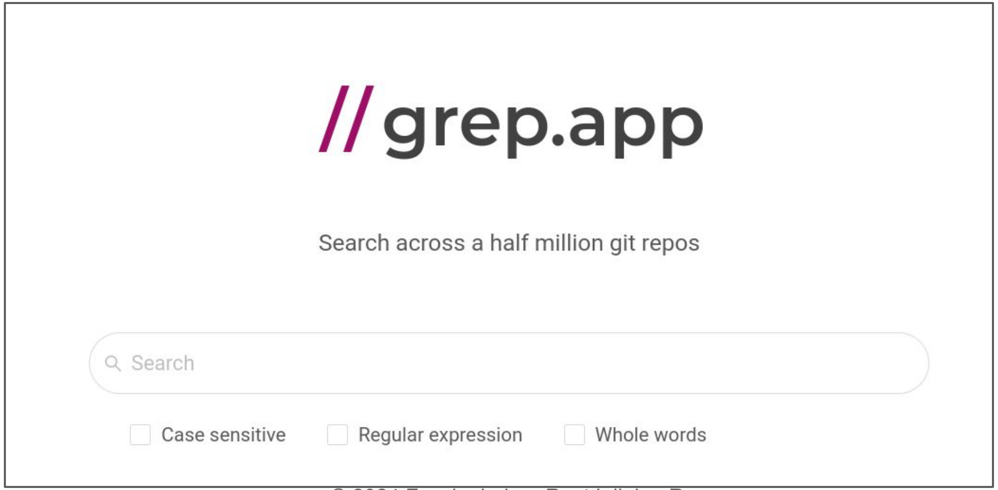
```
          if ((char)bVar2 < -0x40) {
            bVar3 = pbVar8[-3];
            if ((char)bVar3 < -0x40) {
              pbVar7 = pbVar8 + -4;
              uVar13 = bVar3 & 0x3f | (pbVar8[-4] & 7) << 6;
            }
            else {
              pbVar7 = pbVar8 + -3;
              uVar13 = bVar3 & 0xf;
            }
            uVar13 = bVar2 & 0x3f | uVar13 << 6;
          }
          else {
            pbVar7 = pbVar8 + -2;
            uVar13 = bVar2 & 0x1f;
          }
          uVar13 = bVar1 & 0x3f | uVar13 << 6;
          uVar12 = (ulong)uVar13;
          if (uVar13 == 0x110000) goto LAB_00109006;
        }
        else {
          pbVar7 = pbVar8 + -1;
        }
        uVar13 = (uint)uVar12;
      } while ((uVar13 - 9 < 5) || (uVar13 == 0x20));
      if (uVar13 < 0x80) goto LAB_00108ffd;
      uVar6 = (uint)(uVar12 >> 8);
      if (uVar6 < 0x20) {
```

Inlining Recovering

Inlining Recovering: Manual Search

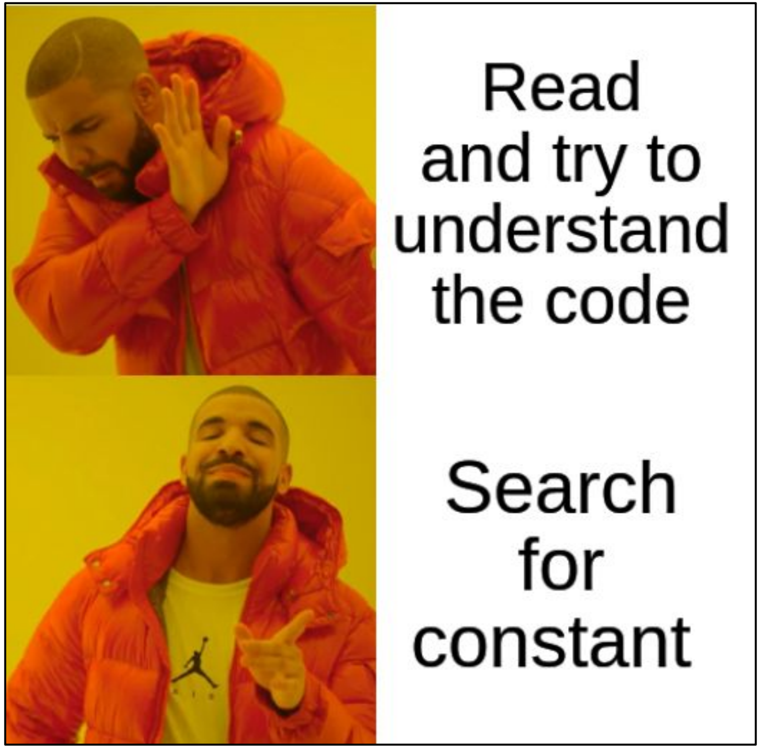
- 2 possibilities : search by **constants** & search by **panic metadata**
- Many libraries have constants
  - For example **md5** hash has **uniques constants.**
  - Use grep.app on all rust github projects, or directly on rust-lang/rust.

- Panic metadata : Directly in **Rust source code**



Recover inlined function: Manual Search

```
  lVar15 = 1;
  if ((char)*(byte *)(lVar16 + -1) < '\0') {
    if (local_b0 == lVar16 + -1) {
      uVar12 = 0;
    }
    else {
      bVar3 = *(byte *)(lVar16 + -2);
      if ((bVar3 & 0xc0) == 0x80) {
        if (local_b0 == lVar16 + -2) {
          uVar12 = 0;
        }
        else {
          bVar2 = *(byte *)(lVar16 + -3);
          if ((bVar2 & 0xc0) == 0x80) {
            if (local_b0 == lVar16 + -3) {
              uVar12 = 0;
            }
            else {
              uVar12 = (*(byte *)(lVar16 + -4) & 7) << 6;
            }
            uVar12 = bVar2 & 0x3f | uVar12;
          }
          else {
            uVar12 = bVar2 & 0xf;
          }
        }
        uVar12 = bVar3 & 0x3f | uVar12 << 6;
      }
      else {
        uVar12 = bVar3 & 0x1f;
      }
    }
    uVar12 = *(byte *)(lVar16 + -1) & 0x3f | uVar12 << 6;
    lVar6 = local_a8._8_8_;
    if (uVar12 == 0x110000) goto LAB_0010683e;
    if ((0x7f < uVar12) && (lVar15 = 2, 0x7ff < uVar12)) {
      lVar15 = 4 - (ulong)(uVar12 < 0x10000);
```



Recover inlined function: Manual Search

- After the read_line : Should be the function **expect**

```
std::io::stdio::Stdin::read_line(&local_a8,&local_60,&local_78);
if (local_a8 != (undefined **)0x0) {
  local_50 = local_a0;
                    /* try { // try from 00109114 to 00109138 has its CatchHandler @ 0010913b */
  core::result::unwrap_failed
            (&DAT_00149011,0x17,&local_50,&PTR_drop_in_place<std_io_error_Error>_0015b0c8,
             &PTR_DAT_0015b0f8);
  do {
    invalidInstructionException();
  } while( true );
}
```

- It calls **unwrap_failed** in **core::result** – > let's check result.rs

Recover inlined function: Manual Search

master rust / library / core / src / result.rs

Code Blame 1980 lines (1885 loc) · 62.2 KB

```rust
    #[inline]
    #[track_caller]
    #[stable(feature = "result_expect", since = "1.4.0")]
    pub fn expect(self, msg: &str) -> T
    where
        E: fmt::Debug,
    {
        match self {
            Ok(t) => t,
            Err(e) => unwrap_failed(msg, &e),
        }
    }
```

All Symbols

function

unwrap_failed

Definition Search

In this file

```
1653 fn unwrap_failed(msg: &str, error: &dyn fmt::Debug) ->
```

7 References Search

In this file

```
1034 Err(e) => unwrap_failed(msg, &e),
1077 Err(e) => unwrap_failed("called `Result::unwrap()` on a
```

FUZZING LABS

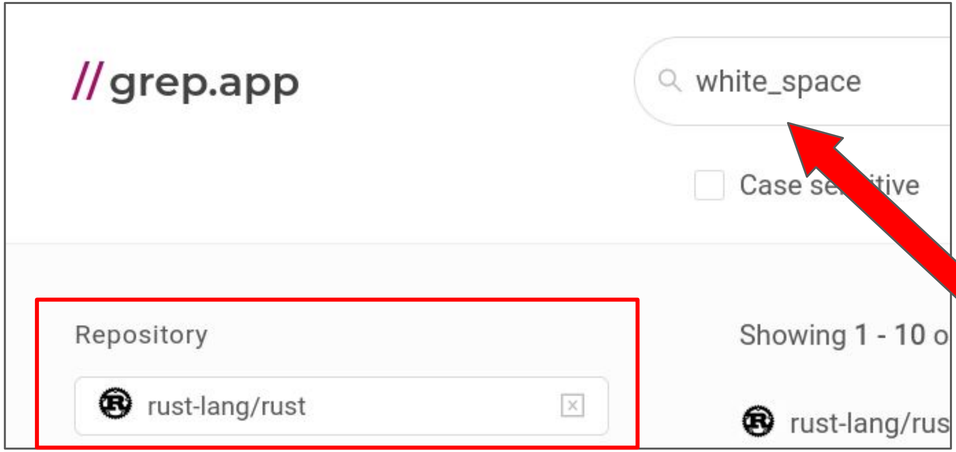
Recover inlined function: Manual Search

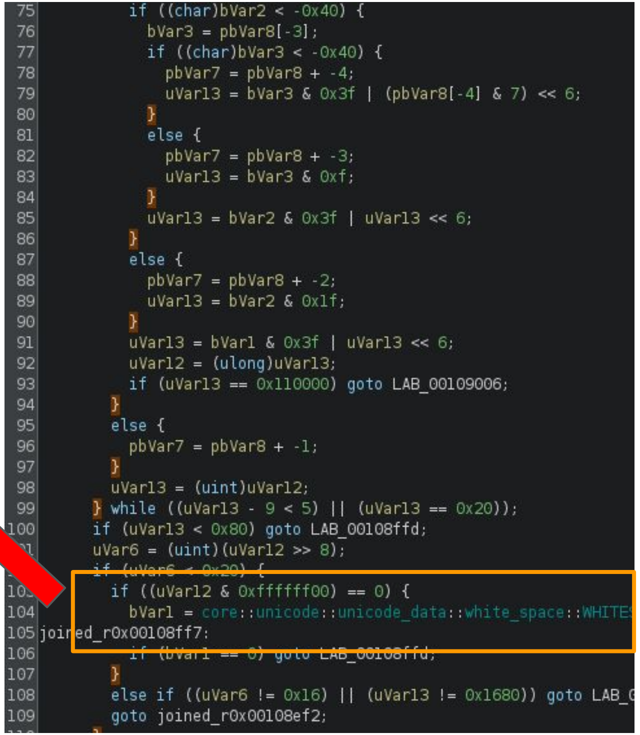
- Next part of the code :
  - No interesting constants to directly find only one reference
  - The **white_space array** is interesting, let's focus on it

```
        else {
          pbVar7 = pbVar8 + -2;
          uVar13 = bVar2 & 0x1f;
        }
        uVar13 = bVar1 & 0x3f | uVar13 << 6;
        uVar12 = (ulong)uVar13;
        if (uVar13 == 0x110000) goto LAB_00109006;
      }
      else {
        pbVar7 = pbVar8 + -1;
      }
      uVar13 = (uint)uVar12;
    } while ((uVar13 - 9 < 5) || (uVar13 == 0x20));
    if (uVar13 < 0x80) goto LAB_00108ffd;
    uVar6 = (uint)(uVar12 >> 8);
    if (uVar6 < 0x20) {
      if ((uVar12 & 0xffffff00) == 0) {
        bVar1 = core::unicode::unicode_data::white_space::WHITESPACE_MAP[uVar12 & 0xff] & 1;
joined r0x00108ff7:
```

Recover inlined function: Manual Search

- Search for **white_space** calls on grep.app





Recover inlined function: Manual Search

- We found **mod.rs** which uses a lot of time **White_Space** -> go deeper
  - We know that we are in mod.rs
  - Find other primitives

- Hypothesis : **trim_end**, **trim_start**, **trim, etc.** -> call **is_whitespace**()
  Which uses the **white_space array**

rust-lang/rust
library/core/src/str/mod.rs

```
    /// 'Whitespace' is defined according to the
    /// Core Property `White_Space`. If you only
    /// instead, use [`split_ascii_whitespace`].

    /// 'Whitespace' is defined according to the
    /// Core Property `White_Space`, which inclu

    /// Core Property `White_Space`, which inclu
```

```
#[inline]
#[must_use = "this returns the trimmed string as a new slice, \
              without modifying the original"]
#[stable(feature = "trim_direction", since = "1.30.0")]
#[cfg_attr(not(test), rustc_diagnostic_item = "str_trim_start")]
pub fn trim_start(&self) -> &str {
    self.trim_start_matches(|c: char| c.is_whitespace())
}
```

Recover inlined function: Search by constant

- Let’s try to find an other primitive with a constant :

```
      if ((uVar6 != 0x30) || (uVar13 != 0x3000)) goto LAB_00108ffd;
    } while( true );
  }
  pbVar11 = (byte *)0x0;
  pbVar8 = local_78;
  pbVar4 = (byte *)0x0;
  do {
    bVar1 = *pbVar8;
    uVar12 = (ulong)bVar1;
    if ((char)bVar1 < '\0') {
      uVar13 = bVar1 & 0x1f;
      if (bVar1 < 0xe0) {
        pbVar9 = pbVar8 + 2;
        uVar12 = (ulong)(uVar13 << 6 | pbVar8[1] & 0x3f);
      }
      else {
        uVar6 = pbVar8[2] & 0x3f | (pbVar8[1] & 0x3f) << 6;
        if (bVar1 < 0xf0) {
          pbVar9 = pbVar8 + 3;
          uVar12 = (ulong)(uVar6 | uVar13 << 0xc);
        }
        else {
          pbVar9 = pbVar8 + 4;
```

Recover inlined function: Search by constant

- Search on **grep.app** the **constant 0xF0** on the Rust Project

- We have **4 references**, let's Check the data/code flow of The First one

Recover inlined function: Search by constant

```rust
fn manual_char_len(s: &str) -> usize {
    let s = s.as_bytes();
    let mut c = 0;
    let mut i = 0;
    let l = s.len();
    while i < l {
        let b = s[i];
        if b < 0x80 {
            i += 1;
        } else if b < 0xe0 {
            i += 2;
        } else if b < 0xf0 {
            i += 3;
        } else {
            i += 4;
        }
        c += 1;
    }
    c
}
```

$\neq$

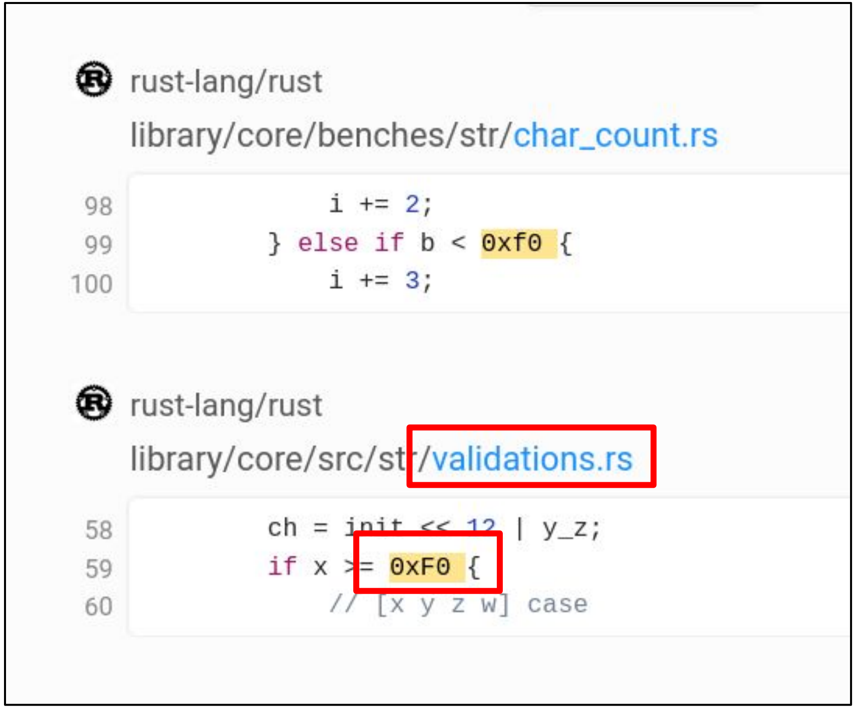
```c
do {
  bVar1 = *pbVar8;
  uVar12 = (ulong)bVar1;
  if ((char)bVar1 < '\0') {
    uVar13 = bVar1 & 0x1f;
    if (bVar1 < 0xe0) {
      pbVar9 = pbVar8 + 2;
      uVar12 = (ulong)(uVar13 << 6 | pbVar8[1] & 0x3f);
    }
    else {
      uVar6 = pbVar8[2] & 0x3f | (pbVar8[1] & 0x3f) << 6;
      if (bVar1 < 0xf0) {
        pbVar9 = pbVar8 + 3;
        uVar12 = (ulong)(uVar6 | uVar13 << 0xc);
      }
      else {
        pbVar9 = pbVar8 + 4;
        uVar13 = pbVar8[3] & 0x3f | uVar6 << 6 | (bVar1 & 7) << 0x12;
        uVar12 = (ulong)uVar13;
        pbVar10 = pbVar4;
        if (uVar13 == 0x110000) goto joined_r0x00108ef2;
      }
    }
  }
```

Recover inlined function: Search by constant

- Search on **grep.app** the **constant 0xF0** on the Rust Project

- The first one isn’t our function
  - Based on the control flow, it’s different
  - Same for the data flow
  - Some part of the code are missing

- let’s try the second one : **validation.rs**



Recover inlined function: Search by constant

```rust
if x >= 0xE0 {
    // [[x y z] w] case
    // 5th bit in 0xE0 .. 0xEF is always clear, so
    // SAFETY: `bytes` produces an UTF-8-like strin
    // so the iterator must produce a value here.
    let z = unsafe { *bytes.next().unwrap_unchecked
    let y_z = utf8_acc_cont_byte((y & CONT_MASK) as
    ch = init << 12 | y_z;
    if x >= 0xF0 {
        // [x y z w] case
        // use only the lower 3 bits of `init`
        // SAFETY: `bytes` produces an UTF-8-like s
        // so the iterator must produce a value her
        let w = unsafe { *bytes.next().unwrap_unche
        ch = (init & 7) << 18 | utf8_acc_cont_byte(
```

```c
do {
  bVar1 = *pbVar8;
  uVar12 = (ulong)bVar1;
  if ((char)bVar1 < '\0') {
    uVar13 = bVar1 & 0x1f;
    if (bVar1 < 0xe0) {
      pbVar9 = pbVar8 + 2;
      uVar12 = (ulong)(uVar13 << 6 | pbVar8[1] & 0x3f);
    }
    else {
      uVar6 = pbVar8[2] & 0x3f | (pbVar8[1] & 0x3f) << 6;
      if (bVar1 < 0xf0) {
        pbVar9 = pbVar8 + 3;
        uVar12 = (ulong)(uVar6 | uVar13 << 0xc);
      }
      else {
        pbVar9 = pbVar8 + 4;
        uVar13 = pbVar8[3] & 0x3f | uVar6 << 6 | (bVar1 & 7) << 0x12;
        uVar12 = (ulong)uVar13;
        pbVar10 = pbVar4;
        if (uVar13 == 0x110000) goto joined_r0x00108ef2;
      }
    }
  }
```

Recover inlined function: Search by constant

- Here is the **call graph to recover** our principal inlined function

rust-lang/rust
library/core/src/str/validations.rs

```rust
        ch = init << 12 | y_z;
        if x >= 0xF0 {
            // [x y z w] case
```

```rust
#[inline]
pub unsafe fn next_code_point<'a, I:
    // Decode UTF-8
    let x = *bytes.next()?;
```

```rust
#[inline]
fn next(&mut self) -> Option<char> {
    // SAFETY: `str` invariant says
    // the resulting `ch` is a valid
    unsafe { next_code_point(&mut se
}
```

- **Next()** was inlined
- Our function uses **Next()** and **White_space**
    - Now we have our different primitives, let's find a function Using those functions

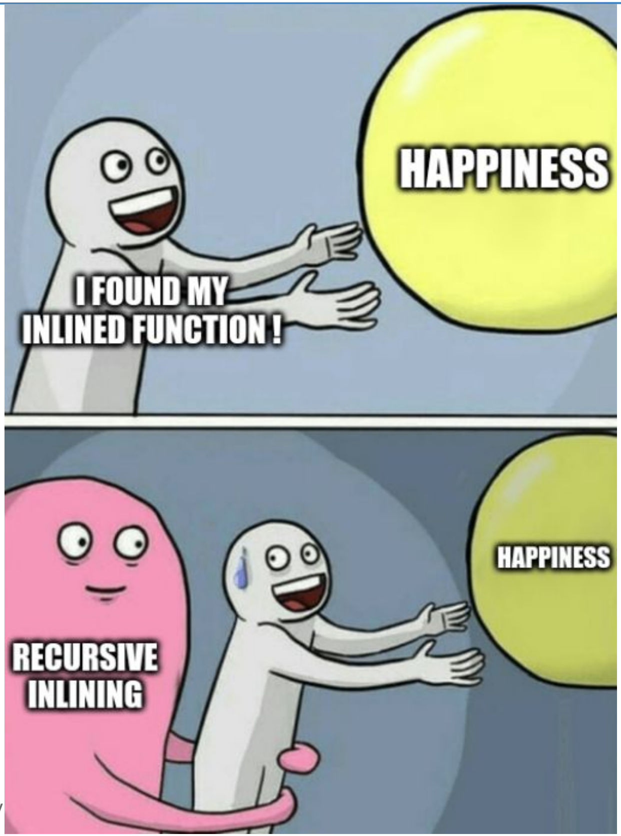
Recover inlined function: Manual Search

- `trim() -> trim_matches() -> next_reject() -> next()`
- We can recover the function **trim()** with this **Call graph**
- **trim()** is the big code block that we have in the Binary, it uses recursive inlining

```
/// Core Property `White_Space`, which includes newlines.
///
/// # Examples
///
/// ```
/// let s = "\n Hello\tworld\t\n";
///
/// assert_eq!("Hello\tworld", s.trim());
/// ```
#[inline]
#[must_use = "this returns the trimmed string as a slice, \
              without modifying the original"]
#[stable(feature = "rust1", since = "1.0.0")]
#[cfg_attr(not(test), rustc_diagnostic_item = "str_trim")]
pub fn trim(&self) -> &str {
    self.trim_matches(|c: char| c.is_whitespace())
}
```

Recover inlined function: Search manually

- Lot of time and work
- Experience needed
  - It's **contextual**
  - No precise **heuristics**
- **Recursive inlining**
  - Inling is often recursive, you need to follow a call graph to find The principal function like we did before



Recover inlined function: Panic metadata

- This **technique** was **found** by Cxiao

- **Rust** binaries **contain paths** to source files and code **locations for panic** occurrences
  - **core::panic::Location** : reveal file paths, line/column numbers
  - This **path** is used for panic printing message
    - thread 'main' panicked at 'Panic message', **src\main.rs**:10:8

- The **path is** located **inside** a **Location structure**
  - **File** : string containing the **path**
  - **Line** : **line of panic** into the native rust function
  - **Col** : **column of panic** into the native rust function

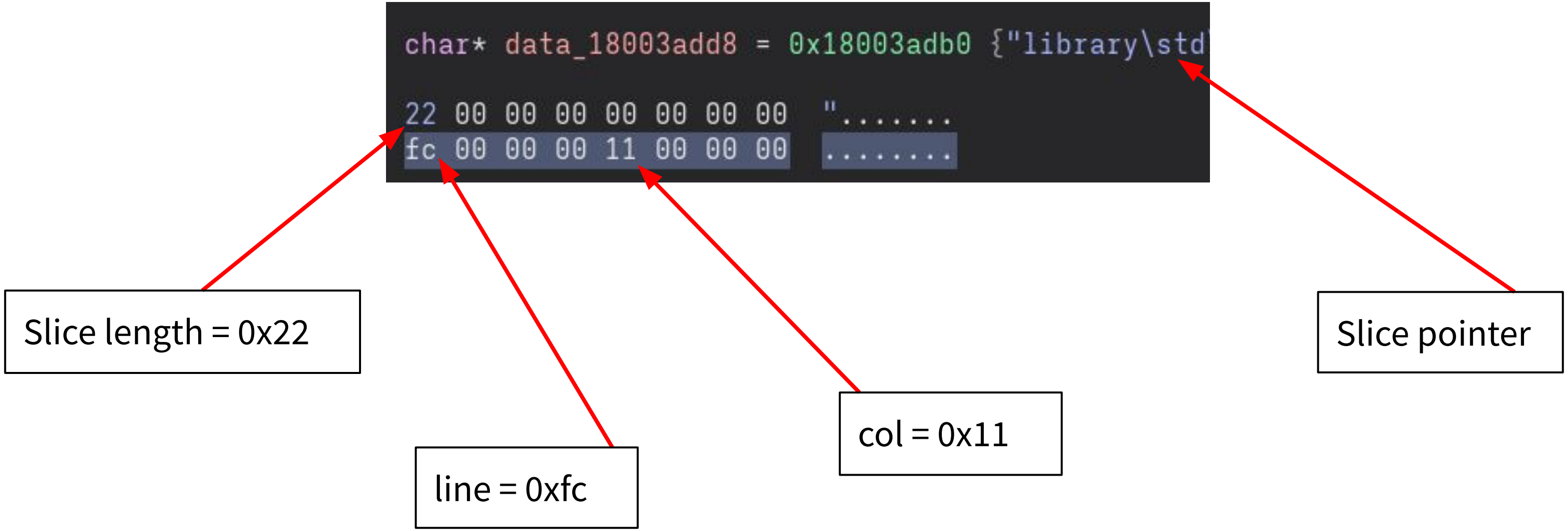
```
pub struct Location<'a> {
    file: &'a str,
    line: u32,
    col: u32,
}
```

- The **line** and **column** field are **metadata**
  - We can use this metadata to get the location of panic into native function.

- **Useful for inlining**
  - Depends on the compilation flags != (panic = abort)

Recover inlined function: Panic metadata

- **Example** with **binary ninja**



Recover inlined function: Panic metadata

- With **binary ninja** we can **create** more **readable structure**

```
struct core::panic::Location panic_location_"library\std\src\sys\windows\mod.rs"
{
    struct RustStringSlice file =
    {
        char* address = data_18003adb0 {"library\std\src\sys\windows\mod.rs"}
        int64_t length = 0x22
    }
    uint32_t line = 0xfc
    uint32_t col = 0x11
}
```

Crab Hacker

Rust Reversing LLM

Crab Hacker: Your Rust Reversing LLM BEST friend

- Crab Hacker
  - GPT bot
  - Domestic Rustacean

- AI tools features
  - **Identifying** and **recovering inlined** functions
  - Give a new pseudo code by **replacing inlined functions** with the call

- README :
  - recover inlined function using **ghidra pseudocode**
  - The bot is a **helper**

Crab Hacker in Action

→

- Original Code ↓

```rust
use std::io;
use std::str;

fn my_function(input: &[u8]) -> Result<&str, str::Utf8Error> {
    str::from_utf8(input)
}

fn main() {
    println!("Enter an input :");

    let mut input = String::new();
    io::stdin().read_line(&mut input).expect("Error reading the input");

    let bytes = input.trim().as_bytes();

    match my_function(bytes) {
        Ok(s) => println!("UTF-8 valide : {}", s),
        Err(e) => println!("UTF-8 invalide, looser : {:?}", e)
    }
}
```

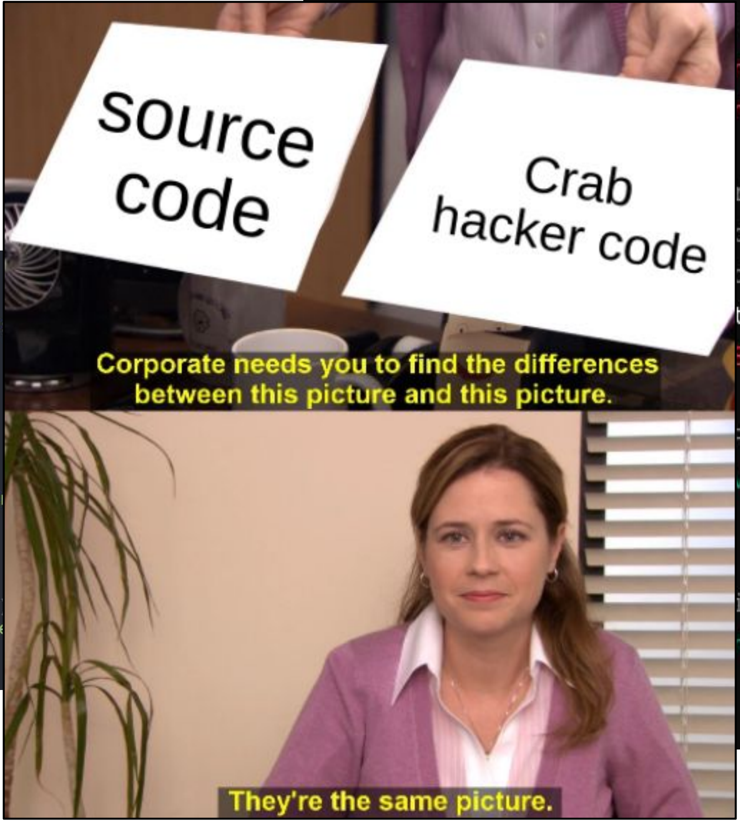
```rust
fn main() {
    // Initial message print, equivalent to a println! macro use
    println!("Enter an UTF-8 string:");

    // Reading from standard input into a string
    let mut input = String::new();
    std::io::stdin().read_line(&mut input).expect("Error while re

    // Trimming the input string and validating UTF-8 encoding
    // These operations are combined into a single step for clari
    // though they involve complex byte-level manipulations as se
    let trimmed_input = input.trim();
    match std::str::from_utf8(trimmed_input.as_bytes()) {
        Ok(valid_str) => {
            // Successful UTF-8 validation
            println!("UTF-8 valid: {}", valid_str);
        },
        Err(e) => {
            // UTF-8 validation failed
            println!("UTF-8 not valid: {:?}", e);
        }
    }
```

Crab Hacker in Action

- Original Code ↓

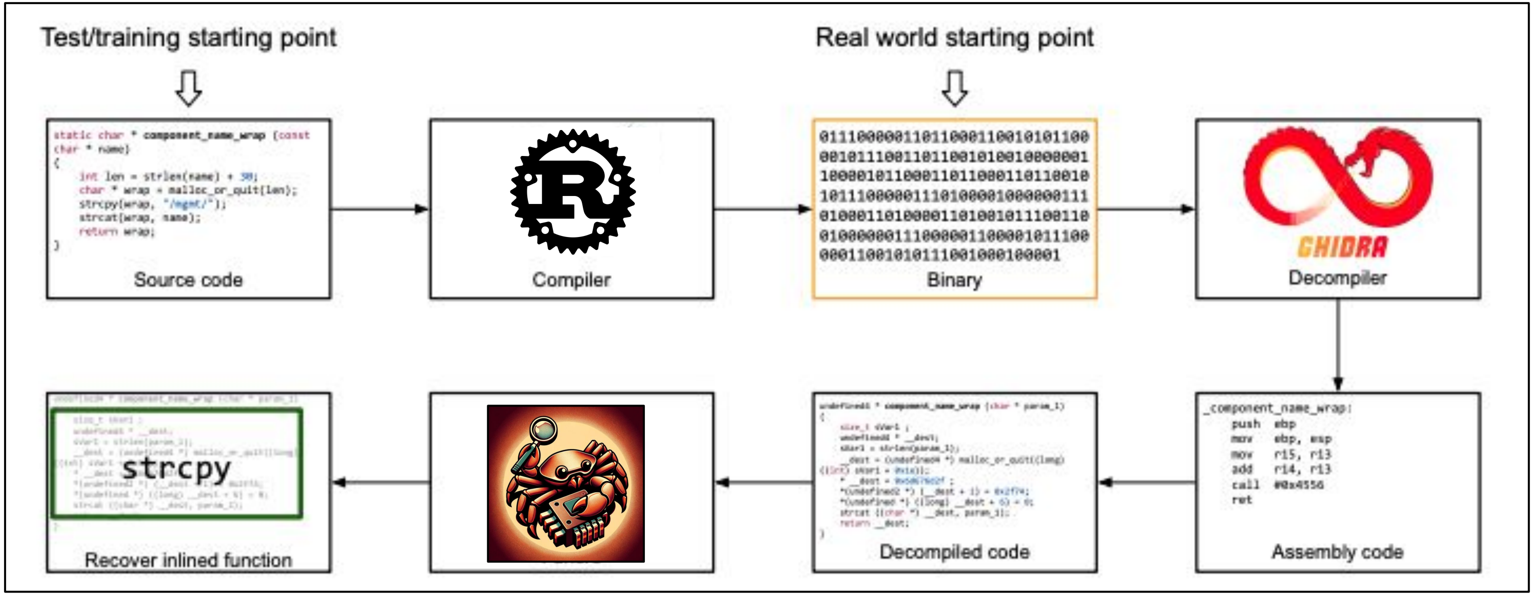
```
use std::io;
use std::str;

fn my_function(input: &[u8]) -> Result<&str, str
    str::from_utf8(input)
}

fn main() {
    println!("Enter an input :");

    let mut input = String::new();
    io::stdin().read_line(&mut input).expect("Er

    let bytes = input.trim().as_bytes();

    match my_function(bytes) {
        Ok(s) => println!("UTF-8 valide : {}", s
        Err(e) => println!("UTF-8 invalide, loose
    }
}
```

```
fn main() {
    // Initial message print, equivalent to a println! macro use
    println!("Enter an UTF-8 string:");

 input into a string
new();
ne(&mut input).expect("Error while re

ring and validating UTF-8 encoding
combined into a single step for clari
omplex byte-level manipulations as se
t.trim();
8(trimmed_input.as_bytes()) {

F-8 validation
valid: {}", valid_str);

ion failed
not valid: {:?}", e);
```



Going Deeper: Create a complete **Rust Dataset**

- Dataset of matching Rust code, assembly, decompiled code to improve **Crab Hacker**
  - **Pattern-oriented** approach adopted by current tools relies on a pattern database
    - Reference paper: Finding Inlined Functions in Optimized Binaries for C++
  - Need to be **manually maintained** to include new pattern, Rust standard libraries and new inlined functions.



Conclusion and Future

Conclusion and future

- Rust reverse engineering is **new** and **hard**
  - **Lack of resources, tools**
- **Active community**
  - [nofix.re]
  - [cxiao.net]
  - [https://github.com/h311d1n3r/Cerberus]
- **Inlining is a problem**
  - No tools
  - It can slow you down
  - Discouraging
- **Crab hacker PoC** could be a solution
  - We need a Dataset
- **Training**
  - [Rust Binary Reverse Engineering by Fuzzinglabs]