Computer System Security

Lab Assignment 3 Report

Sonam Ghatode

Arjun Katneni

11.19.2019

Index

Variable Bindings :

1. “variables1.rs”:

```
1  // variables1.rs
2  // Make me compile! Scroll down for hints :)
3
4  fn main() {
5      x = 5;
6      println!("x has the value {}", x);
7  }
8
9
10
11
12
13
14
15
16
17
18
19
20
```

E

```
   Compiling playground v0.0.1 (/playground)
error[E0425]: cannot find value `x` in this scope
 --> src/main.rs:5:5
  |
5 |     x = 5;
  |     ^ not found in this scope

error[E0425]: cannot find value `x` in this scope
 --> src/main.rs:6:36
  |
6 |     println!("x has the value {}", x);
  |                                    ^ not found in this scope

error: aborting due to 2 previous errors

For more information about this error, try `rustc --explain E0425`.
```

As we can see in the error that variable was not found in the scope of *main()* function. So to rectify this error, we added ***let*** before ***x = 5;*** to declare the variable ***x***. Following is the screenshot for the same.

```rust
// variables1.rs
// Make me compile! Scroll down for hints :)

fn main() {
    let x = 5;
    println!("x has the value {}", x);
}
```

```
   Compiling playground v0.0.1 (/playground)
    Finished dev [unoptimized + debuginfo] target(s) in 0.46s
     Running `target/debug/playground`
```

```
x has the value 5
```

2. “variables2.rs”:

In this program, variable ***x*** is not given any type. Rust do not allow a variable to be declared without a type to ensure type safety. Following is a screenshot that shows the error in the program while we try to compile the program without giving type to variable ***x***.

```
// variables2.rs
// Make me compile! Scroll down for hints :)

fn main() {
    let x;
    if x == 10 {
        println!("Ten!");
    } else {
        println!("Not ten!");
    }
}
```

```
   Compiling playground v0.0.1 (/playground)
error[E0282]: type annotations needed
 --> src/main.rs:5:9
  |
5 |     let x;
  |         ^ consider giving `x` a type

error: aborting due to previous error

For more information about this error, try `rustc --explain E0282`.
error: could not compile `playground`.

To learn more, run the command again with --verbose.
```

To rectify this error, we gave a type to the variable ***x***. By assigning a value to a variable at the time of declaration binds it to the type of data assigned at the time of declaration. And in memory, if there is data value already stored because of earlier memory mapping, type mismatch or use-after-free can happen, which has serious security implications. Following is the screenshot for the same:

```
1  // variables2.rs
2  // Make me compile! Scroll down for hints :)
3
4  fn main() {
5      let x = 0;
6      if x == 10 {
7          println!("Ten!");
8      } else {
9          println!("Not ten!");
10     }
11 }
12
13
14
15
16
17
18
19
20
21
```

```
   Compiling playground v0.0.1 (/playground)
    Finished dev [unoptimized + debuginfo] target(s) in 0.81s
     Running `target/debug/playground`
```

```
Not ten!
```

3. "variables3.rs":

In the program shown in the screenshot below, the variable ***x*** is declared in line 5 as immutable variable and is being mutated in line number 7. This is not allowed in Rust Programming Language. One data can have only one owner and vice versa. The program below violates this condition. The security implication here is that if a variable's value can be changed in it's scope, it's considered vulnerable from a security point of view.

```rust
// variables3.rs
// Make me compile! Scroll down for hints :)

fn main() {
    let x = 3;
    println!("Number {}", x);
    x = 5;
    println!("Number {}", x);
}
```

```
   Compiling playground v0.0.1 (/playground)
error[E0384]: cannot assign twice to immutable variable `x`
 --> src/main.rs:7:5
  |
5 |     let x = 3;
  |         -
  |         |
  |         first assignment to `x`
  |         help: make this binding mutable: `mut x`
6 |     println!("Number {}", x);
7 |     x = 5;
  |     ^^^^^ cannot assign twice to immutable variable

error: aborting due to previous error

For more information about this error, try `rustc --explain E0384`.
```

For ***x*** to be modified later in the program, it has to be declared mutable. Following is the screenshot for the corrected version of the program:

```rust
// variables3.rs
// Make me compile! Scroll down for hints :)

fn main() {
    let mut x = 3;
    println!("Number {}", x);
    x = 5;
    println!("Number {}", x);
}
```

```
   Compiling playground v0.0.1 (/playground)
    Finished dev [unoptimized + debuginfo] target(s) in 2.33s
     Running `target/debug/playground`
```

```
Number 3
Number 5
```

4. “variables4.rs”:

In the program ***variables4.rs***, variable ***x*** is not initialized with any value, making it basically typeless, which is against the convention followed in Rust Programming Language.

```rust
// variables4.rs
// Make me compile! Scroll down for hints :)

fn main() {
    let x: i32;
    println!("Number {}", x);
}
```

```
   Compiling playground v0.0.1 (/playground)
error[E0381]: borrow of possibly-uninitialized variable: `x`
 --> src/main.rs:6:27
  |
6 |     println!("Number {}", x);
  |                           ^ use of possibly-uninitialized `x`

error: aborting due to previous error

For more information about this error, try `rustc --explain E0381`.
error: could not compile `playground`.

To learn more, run the command again with --verbose.
```

Uninitialized variable when stored in a memory can point to a memory location which earlier was used and then freed, i.e. garbage value or some arbitrary or type mismatched data. This can have a security implication same as use-after-free situation has.

To remove the error encountered in the program, we initialized variable ***x*** with a value and compiled again. Following is the screenshot for the same:

```rust
// variables4.rs
// Make me compile! Scroll down for hints :)

fn main() {
    let x: i32 = 20;
    println!("Number {}", x);
}
```

```
   Compiling playground v0.0.1 (/playground)
    Finished dev [unoptimized + debuginfo] target(s) in 0.47s
     Running `target/debug/playground`
```

```
Number 20
```

Functions :

1. “functions1.rs”:

In the program ***functions1.rs***, function ***call_me()*** is called from function ***main()***, but is never defined:

```
1  // functions1.rs
2  // Make me compile! Scroll down for hints :)
3
4  fn main() {
5      call_me();
6  }
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

```
   Compiling playground v0.0.1 (/playground)
error[E0425]: cannot find function `call_me` in this scope
 --> src/main.rs:5:5
  |
5 |     call_me();
  |     ^^^^^^^ not found in this scope

error: aborting due to previous error

For more information about this error, try `rustc --explain E0425`.
error: could not compile `playground`.

To learn more, run the command again with --verbose.
```

To remove the error, we defined a function ***call_me()***, which just prints the line ***I'm in call_me()***. Following is the screenshot for the same:

```rust
// functions1.rs
// Make me compile! Scroll down for hints :)

fn main() {
    call_me();
}

fn call_me() {
    println!("I'm in call_me()")
}
```

```
   Compiling playground v0.0.1 (/playground)
    Finished dev [unoptimized + debuginfo] target(s) in 1.65s
     Running `target/debug/playground`
```

```
I'm in call_me()
```

2. “functions2.rs”:

The program in the screenshot below passes value ***3*** as a parameter of function ***call_me()***, but in the function definition, it is never specified whether the ownership of the value is transferred to the function ***call_me()*** argument ***num*** or value is copied to the function argument ***num***:

```rust
// functions2.rs
// Make me compile! Scroll down for hints :)

fn main() {
    call_me(3);
}

fn call_me(num) {
    for i in 0..num {
        println!("Ring! Call number {}", i + 1);
    }
}
```

```
   Compiling playground v0.0.1 (/playground)
error: expected one of `:`, `@`, or `|`, found `)`
 --> src/main.rs:8:15
  |
8 | fn call_me(num) {
  |               ^ expected one of `:`, `@`, or `|` here
  |
  = note: anonymous parameters are removed in the 2018 edition (
help: if this was a parameter name, give it a type
  |
8 | fn call_me(num: TypeName) {
  |            ^^^^^^^^^^^^^
help: if this is a type, explicitly ignore the parameter name
  |
8 | fn call_me(_: num) {
  |            ^^^^^^
```

Security implication in this case is that the value that will be passed to the function can be modified further without ***num*** being the owner of the data, which is against the security principle explained in rust programming. We mentioned ***: i32*** after function argument ***num***, specifying that the value is just copied to ***num***, which removed the error:

```rust
// functions2.rs
// Make me compile! Scroll down for hints :)

fn main() {
    call_me(3);
}

fn call_me(num: i32) {
    for i in 0..num {
        println!("Ring! Call number {}", i + 1);
    }
}
```

```
   Compiling playground v0.0.1 (/playground)
    Finished dev [unoptimized + debuginfo] target(s) in 0.79s
     Running `target/debug/playground`
```

```
Ring! Call number 1
Ring! Call number 2
Ring! Call number 3
```

3. “functions3.rs”:

As explained in the error description in the screenshot below, function ***call_me()*** takes a parameter but none was passed when it was called from ***main()***:

```
1  // functions3.rs
2  // Make me compile! Scroll down for hints :)
3
4  fn main() {
5      call_me();
6  }
7
8  fn call_me(num: i32) {
9      for i in 0..num {
10         println!("Ring! Call number {}", i + 1);
11     }
12 }
13
14
15
16
17
18
19
20
```

Exec

Sta

```
   Compiling playground v0.0.1 (/playground)
error[E0061]: this function takes 1 parameter but 0 parameters were supplied
 --> src/main.rs:5:5
  |
5 |     call_me();
  |     ^^^^^^^^^ expected 1 parameter
...
8 | fn call_me(num: i32) {
  | -------------------- defined here

error: aborting due to previous error

For more information about this error, try `rustc --explain E0061`.
error: could not compile `playground`.

To learn more, run the command again with --verbose.
```

To remove this error, we passed a value in the function call of function ***call_me()***. The program does not satisfy the function syntax. The called function is not defined in the program and hence should not be allowed to make the call. This satisfied the requirement of a parameter to be passed during function call. Following is the screenshot for the same:

```rust
// functions3.rs
// Make me compile! Scroll down for hints :)

fn main() {
    call_me(10);
}

fn call_me(num: i32) {
    for i in 0..num {
        println!("Ring! Call number {}", i + 1);
    }
}
```

```
   Compiling playground v0.0.1 (/playground)
    Finished dev [unoptimized + debuginfo] target(s) in 0.61s
     Running `target/debug/playground`
```

```
Ring! Call number 1
Ring! Call number 2
Ring! Call number 3
Ring! Call number 4
Ring! Call number 5
Ring! Call number 6
Ring! Call number 7
Ring! Call number 8
Ring! Call number 9
Ring! Call number 10
```

4. “functions4.rs”:

In function definition of function ***sale_price***, no return type is specified and therefore, the program compilation threw error and got aborted. Following is the screenshot for the same:

```
6
7  fn main() {
8      let original_price = 51;
9      println!("Your sale price is {}", sale_price(original_price));
10 }
11
12 fn sale_price(price: i32) -> {
13     if is_even(price) {
14         price - 10
15     } else {
16         price - 3
17     }
18 }
19
20 fn is_even(num: i32) -> bool {
21     num % 2 == 0
22 }
23
24
25
```

```
   Compiling playground v0.0.1 (/playground)
error: expected type, found `{`
  --> src/main.rs:12:30
   |
12 | fn sale_price(price: i32) -> {
   |                              ^ expected type

error: aborting due to previous error

error: could not compile `playground`.

To learn more, run the command again with --verbose.
```

If the return type is specified, the compilation will be successful. The program does not follow the syntax of the language, possible security implication is any value of any type can be returned in this case. Following is the screenshot for the corrected program and it's successful compilation:

```
6
7  fn main() {
8      let original_price = 51;
9      println!("Your sale price is {}", sale_price(original_price));
10 }
11
12 fn sale_price(price: i32) -> i32{
13     if is_even(price) {
14         price - 10
15     } else {
16         price - 3
17     }
18 }
19
20 fn is_even(num: i32) -> bool {
21     num % 2 == 0
22 }
23
24
25
```

```
   Compiling playground v0.0.1 (/playground)
    Finished dev [unoptimized + debuginfo] target(s) in 0.43s
     Running `target/debug/playground`
```

```
Your sale price is 48
```

5. “functions5.rs”:

In the program below, the return type is specified as ***i32***, however in the function body, return statement have semicolon, making the function as non-returnable. Following is the screenshot for the same:

```rust
// functions5.rs
// Make me compile! Scroll down for hints :)

fn main() {
    let answer = square(3);
    println!("The answer is {}", answer);
}

fn square(num: i32) -> i32 {
    num * num;
}
```

Executi

Stand

```
   Compiling playground v0.0.1 (/playground)
error[E0308]: mismatched types
  --> src/main.rs:9:24
   |
9  | fn square(num: i32) -> i32 {
   |    ------              ^^^ expected i32, found ()
   |    |
   |    implicitly returns `()` as its body has no tail or `return` expression
10 |     num * num;
   |               - help: consider removing this semicolon
   |
   = note: expected type `i32`
              found type `()`

error: aborting due to previous error
```

To remove this error, we just removed the semicolon from the statement ***num*num;*** and compiled the program to get an error-free code. Security implication in this case is nothing, basically the problem is program didn't follow the syntax of the language. Following is the screenshot for the same:

```rust
// functions5.rs
// Make me compile! Scroll down for hints :)

fn main() {
    let answer = square(3);
    println!("The answer is {}", answer);
}

fn square(num: i32) -> i32 {
    num * num
}
```

```
   Compiling playground v0.0.1 (/playground)
    Finished dev [unoptimized + debuginfo] target(s) in 0.70s
     Running `target/debug/playground`
```

```
The answer is 9
```

Primitive Types :

1. “primitive_types1.rs”:

In the program below, there’s missing declaration after ***let*** and hence not in proper syntax. To make the program follow the syntax, we declared the variable used in line next to it i.e. ***is_evening***. The usage of a variable that is not even found in the current scope should never be allowed.

```
fn main() {
    // Booleans (`bool`)

    let is_morning = true;
    if is_morning {
        println!("Good morning!");
    }

    let
    if is_evening {
        println!("Good evening!");
    }
}
```

```
   Compiling playground v0.0.1 (/playground)
error: expected identifier, found keyword `if`
  --> src/main.rs:14:5
   |
14 |     if is_evening {
   |     ^^ expected identifier, found keyword
help: you can escape reserved keywords to use them as identifiers
   |
14 |     r#if is_evening {
   |     ^^^^

error: expected one of `:`, `;`, `=`, `@`, or `|`, found `is_evening`
  --> src/main.rs:14:8
   |
14 |     if is_evening {
   |        ^^^^^^^^^^ expected one of `:`, `;`, `=`, `@`, or `|` here
```

Therefore, to rectify the error encountered in the ***main()***, we declared ***is_evening*** variable and initialized it with a value. Following is the screenshot for the same:

```rust

fn main() {

    let is_morning = true;
    if is_morning {
        println!("Good morning!");
    }

    let is_evening = false;
    if is_evening {
        println!("Good evening!");
    }
}

```

```
   Compiling playground v0.0.1 (/playground)
    Finished dev [unoptimized + debuginfo] target(s) in 0.43s
     Running `target/debug/playground`
```

```
Good morning!
```

2. “primitive_types2.rs”:

As variable ***your_character*** is never declared and initialized in the scope of ***main()***, the program threw an error that ***your_character*** is never found in the scope. The syntax of languages like Rust will never allow this kind of variable usage. Following is a screenshot of the error we got when we tried to compile the program as it was:

```
fn main() {
    let my_first_initial = 'C';
    if my_first_initial.is_alphabetic() {
        println!("Alphabetical!");
    } else if my_first_initial.is_numeric() {
        println!("Numerical!");
    } else {
        println!("Neither alphabetic nor numeric!");
    }

    let
    if your_character.is_alphabetic() {
        println!("Alphabetical!");
    } else if your_character.is_numeric() {
        println!("Numerical!");
    } else {
        println!("Neither alphabetic nor numeric!");
    }
}
```

Exe

St

```
   Compiling playground v0.0.1 (/playground)
error: expected identifier, found keyword `if`
  --> src/main.rs:12:5
   |
12 |     if your_character.is_alphabetic() {
   |     ^^ expected identifier, found keyword
help: you can escape reserved keywords to use them as identifiers
   |
12 |     r#if your_character.is_alphabetic() {
   |     ^^^^

error: expected one of `:`, `;`, `=`, `@`, or `|`, found `your_character`
  --> src/main.rs:12:8
   |
12 |     if your_character.is_alphabetic() {
   |        ^^^^^^^^^^^^^^^ expected one of `:`, `;`, `=`, `@`, or `|` here
```

To remove the error encountered in the program, we declared and initialized the variable ***your_character***. Following is the screenshot for the corrected program and it's output:

```
fn main() {
    let my_first_initial = 'C';
    if my_first_initial.is_alphabetic() {
        println!("Alphabetical!");
    } else if my_first_initial.is_numeric() {
        println!("Numerical!");
    } else {
        println!("Neither alphabetic nor numeric!");
    }

    let your_character = '#';
    if your_character.is_alphabetic() {
        println!("Alphabetical!");
    } else if your_character.is_numeric() {
        println!("Numerical!");
    } else {
        println!("Neither alphabetic nor numeric!");
    }
}
```

```
   Compiling playground v0.0.1 (/playground)
    Finished dev [unoptimized + debuginfo] target(s) in 0.41s
     Running `target/debug/playground`
```

```
Alphabetical!
Neither alphabetic nor numeric!
```

3. "primitive_types3.rs":

In the code below, variable ***a*** is never assigned a value. An uninitialized variable, as discussed above can have serious implications as far as the security of a language is concerned. Following is a screenshot of the error we got while compiling this code:

```
fn main() {
    let a = ???

    if a.len() >= 100 {
        println!("Wow, that's a big array!");
    } else {
        println!("Meh, I eat arrays like that for breakfast.");
    }
}
```

```
   Compiling playground v0.0.1 (/playground)
error: expected expression, found `?`
 --> src/main.rs:2:13
  |
2 |     let a = ???
  |             ^ expected expression

error: aborting due to previous error

error: could not compile `playground`.

To learn more, run the command again with --verbose.
```

We declared the array ***a*** with 10 elements and got the following output:

```
fn main() {
    let a = ['A';10];

    if a.len() >= 100 {
        println!("Wow, that's a big array!");
    } else {
        println!("Meh, I eat arrays like that for breakfast.");
    }
}
```

```
   Compiling playground v0.0.1 (/playground)
    Finished dev [unoptimized + debuginfo] target(s) in 0.79s
     Running `target/debug/playground`
```

```
Meh, I eat arrays like that for breakfast.
```

4. “primitive_types4.rs”:

In the screenshot below, ***nice_slice*** is left uninitialized, the reason why program threw error. Security implication here is that if ***nice_slice*** does not have type that matches the conditional statement below, the comparison should never be allowed in the first place. Another security implication here is that variable ***a*** is declared and initialized but never used in the program ahead. It should never be allowed in a memory and type safe program. It is basically wasting the program memory.

```
1
2  fn main() {
3      let a = [1, 2, 3, 4, 5];
4
5      let nice_slice = ???
6
7      if nice_slice == [2, 3, 4] {
8          println!("Nice slice!");
9      } else {
10         println!("Not quite what I was expecting... I see: {:?}", nice_slice);
11     }
12 }
13
14
15
16
17
18
19
20
```

Execution

Standard Err

```
   Compiling playground v0.0.1 (/playground)
error: expected expression, found `?`
 --> src/main.rs:5:22
  |
5 |     let nice_slice = ???
  |                      ^ expected expression

error: aborting due to previous error

error: could not compile `playground`.

To learn more, run the command again with --verbose.
```

To remove the error, we declared the ***nice_slice*** variable with the value conforming to the type that could be matched in the statement below it. And since variable ***a*** is never used in the program again, we prefixed it with an ***underscore*** sign to suppress the warning of the unused variable. Following is the screenshot for the rectified program:

```
1
2  fn main() {
3      let _a = [1, 2, 3, 4, 5];
4
5      let nice_slice = [2,4,5];
6
7      if nice_slice == [2, 3, 4] {
8          println!("Nice slice!");
9      } else {
10         println!("Not quite what I was expecting... I see: {:?}", nice_slice);
11     }
12 }
13
14
15
16
17
18
19
20
```

Execution

Standard Er

```
   Compiling playground v0.0.1 (/playground)
    Finished dev [unoptimized + debuginfo] target(s) in 0.46s
     Running `target/debug/playground`
```

Standard Out

```
Not quite what I was expecting... I see: [2, 4, 5]
```

5. "primitive_types5.rs":

In the program below, variable ***cat*** is declared and initialized with a tuple data. Therefore a pattern should also be declared for the same. Tuples have a fixed length: once declared, they cannot grow or shrink in size. Since it was not declared, the program threw the error. Following is the screenshot for the same:

```
1
2
3  fn main() {
4      let cat = ("Furry McFurson", 3.5);
5      let  = cat;
6
7      println!("{} is {} years old.", name, age);
8  }
9
10
11
12
13
14
15
16
17
18
19
20
```

```
   Compiling playground v0.0.1 (/playground)
error: expected pattern, found `=`
 --> src/main.rs:5:10
  |
5 |     let  = cat;
  |          ^ expected pattern

error: aborting due to previous error

error: could not compile `playground`.

To learn more, run the command again with --verbose.
```

Following is the corrected code with the tuple definition for variable ***cat***:

```
1
2
3  fn main() {
4      let cat = ("Furry McFurson", 3.5);
5      let (name, age) = cat;
6
7      println!("{} is {} years old.", name, age);
8  }
9
10
11
12
13
14
15
16
17
18
19
20
```

```
   Compiling playground v0.0.1 (/playground)
    Finished dev [unoptimized + debuginfo] target(s) in 0.43s
     Running `target/debug/playground`
```

```
Furry McFurson is 3.5 years old.
```

6. “primitive_types6.rs”:

In the program below, the tuple data is declared, but the way to access the data in the following print statement is missing. To access the tuple data, “.” operator is used. Following is a screenshot of the error encountered while trying to compile the code:

```
1
2
3  fn main() {
4      let numbers = (1, 2, 3);
5      println!("The second number is {}", ???);
6  }
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

```
   Compiling playground v0.0.1 (/playground)
error: expected expression, found `?`
 --> src/main.rs:5:41
  |
5 |     println!("The second number is {}", ???);
  |                                         ^ expected expression

error: aborting due to previous error

error: could not compile `playground`.

To learn more, run the command again with --verbose.
```

To access the tuple data, we modified the program as follows:

```
1
2
3  fn main() {
4      let numbers = (1, 2, 3);
5      println!("The second number is {}", numbers.1);
6  }
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

```
   Compiling playground v0.0.1 (/playground)
    Finished dev [unoptimized + debuginfo] target(s) in 0.45s
     Running `target/debug/playground`
```

```
The second number is 2
```

Strings :

1. “strings1.rs”:

The String type, which is provided by Rust’s standard library rather than coded into the core language, is a growable, mutable, owned, UTF-8 encoded string type. Rust has only one string type in the core language, which is the string slice str that is usually seen in its borrowed form &str.

```
1
2  fn main() {
3      let answer = current_favorite_color();
4      println!("My current favorite color is {}", answer);
5  }
6
7  fn current_favorite_color() -> String {
8      "blue"
9  }
10
11
12
13
14
15
16
17
18
19
20
```

```
   Compiling playground v0.0.1 (/playground)
error[E0308]: mismatched types
 --> src/main.rs:8:5
  |
7 | fn current_favorite_color() -> String {
  |                                ------ expected `std::string::String`
8 |     "blue"
  |     ^^^^^^
  |     |
  |     expected struct `std::string::String`, found reference
  |     help: try using a conversion method: `"blue".to_string()`
  |
  = note: expected type `std::string::String`
             found type `&'static str`

error: aborting due to previous error
```

Function ***current_favorite_color()*** expected to return a String type return value, but the statement ***"blue"*** does not explicitly states that it's a String. To solve this problem, we did explicit type-casting for the same and got an error-free code. Following is the screenshot for the same:

```
1
2  fn main() {
3      let answer = current_favorite_color();
4      println!("My current favorite color is {}", answer);
5  }
6
7  fn current_favorite_color() -> String {
8      "blue".to_string()
9  }
10
11
12
13
14
15
16
17
18
19
20
```

```
   Compiling playground v0.0.1 (/playground)
    Finished dev [unoptimized + debuginfo] target(s) in 0.73s
     Running `target/debug/playground`
```

```
My current favorite color is blue
```

2. “strings2.rs”:

In the program below, function ***is_a_color_word()*** is trying to access or take ownership of the String type variable ***word***, which by language syntax is not allowed, therefore the program throws error when we try to run this program. Following is a screenshot of the error encountered:

```
fn main() {
    let word = String::from("green");
    if is_a_color_word(word) {
        println!("That is a color word I know!");
    } else {
        println!("That is not a color word I know.");
    }
}

fn is_a_color_word(attempt: &str) -> bool {
    attempt == "green" || attempt == "blue" || attempt == "red"
}
```

Exe

Sta

```
   Compiling playground v0.0.1 (/playground)
error[E0308]: mismatched types
 --> src/main.rs:4:24
  |
4 |     if is_a_color_word(word) {
  |                        ^^^^
  |                        |
  |                        expected &str, found struct `std::string::String`
  |                        help: consider borrowing here: `&word`
  |
  = note: expected type `&str`
             found type `std::string::String`

error: aborting due to previous error

For more information about this error, try `rustc --explain E0308`.
```

The security implication here is the fact that String type is most vulnerable while considering security. As we have seen in almost all buffer overflow attacks, the mutability of the String data type is exploited, so to get rid of this vulnerability associated with the mutability of String data types, rust made them immutable by default, and if a mutable string is needed, it has to be declared mutable and borrowed properly to carry on with the program. Following is the screenshot after making ***is_a_color_word()*** borrow ***word***:

```
1
2  fn main() {
3      let word = String::from("green");
4      if is_a_color_word(&word) {
5          println!("That is a color word I know!");
6      } else {
7          println!("That is not a color word I know.");
8      }
9  }
10
11 fn is_a_color_word(attempt: &str) -> bool {
12     attempt == "green" || attempt == "blue" || attempt == "red"
13 }
14
15
16
17
18
19
20
```

```
   Compiling playground v0.0.1 (/playground)
    Finished dev [unoptimized + debuginfo] target(s) in 0.56s
     Running `target/debug/playground`
```

```
That is a color word I know!
```

3. “strings3.rs”:

In the program below, two functions are defined, namely ***string_slice()*** and ***string*** which takes string slice and string, respectively, as input. We had to recognize which statement should be calling which function as their input differ. Following is a screenshot of the program before making the changes:

```
fn string_slice(arg: &str) { println!("{}", arg); }
fn string(arg: String) { println!("{}", arg); }

fn main() {
    ("blue");
    ("red".to_string());
    (String::from("hi"));
    ("rust is fun!".to_owned());
    ("nice weather".into());
    (format!("Interpolation {}", "Station"));
    (&String::from("abc")[0..1]);
    ("  hello there ".trim());
    ("Happy Monday!".to_string().replace("Mon", "Tues"));
    ("mY sHiFt KeY iS sTiCkY".to_lowercase());
}
```

```
   Compiling playground v0.0.1 (/playground)
error[E0282]: type annotations needed
  --> src/main.rs:10:21
   |
10 |     ("nice weather".into());
   |                     ^^^^ cannot infer type for `T`

error: aborting due to previous error

For more information about this error, try `rustc --explain E0282`.
error: could not compile `playground`.

To learn more, run the command again with --verbose.
```

The string slice is immutable and no operation apart from accessing its elements can be performed on a slice. But String can be made mutable and operations can be performed on the same. Keeping that in mind, we edited the code to come up with the following solution:

```
1
2  fn string_slice(arg: &str) { println!("{}", arg); }
3  fn string(arg: String) { println!("{}", arg); }
4
5  fn main() {
6      string_slice("blue");
7      string("red".to_string());
8      string(String::from("hi"));
9      string("rust is fun!".to_owned());
10     string("nice weather".into());
11     string(format!("Interpolation {}", "Station"));
12     string_slice(&String::from("abc")[0..1]);
13     string_slice("  hello there ".trim());
14     string("Happy Monday!".to_string().replace("Mon", "Tues"));
15     string("mY sHiFt KeY iS sTiCkY".to_lowercase());
16 }
17
```

```
   Compiling playground v0.0.1 (/playground)
    Finished dev [unoptimized + debuginfo] target(s) in 0.92s
     Running `target/debug/playground`
```

```
blue
red
hi
rust is fun!
nice weather
Interpolation Station
a
hello there
Happy Tuesday!
my shift key is sticky
```

Move Semantics :

1. “move_semantics1.rs”:

In the code below, statement in line 8 is trying to modify a variable value that is not mutable. In Rust, by default any variable declared can never be mutated in their lifetime unless declared mutated otherwise. If this was not the case and a variable is allowed to be mutated multiple times, if another variable or function is trying to access it during the time of mutation, it can result in non-integral values. Following is a screenshot of the error:

```
pub fn main() {
    let vec0 = Vec::new();

    let vec1 = fill_vec(vec0);

    println!("{} has length {} content `{:?}`", "vec1", vec1.len(), vec1);

    vec1.push(88);

    println!("{} has length {} content `{:?}`", "vec1", vec1.len(), vec1);

}

fn fill_vec(vec: Vec<i32>) -> Vec<i32> {
    let mut vec = vec;

    vec.push(22);
    vec.push(44);
    vec.push(66);

```

Execution

Standar

```
   Compiling playground v0.0.1 (/playground)
error[E0596]: cannot borrow `vec1` as mutable, as it is not declared as mutable
 --> src/main.rs:8:5
  |
4 |     let vec1 = fill_vec(vec0);
  |         ---- help: consider changing this to be mutable: `mut vec1`
...
8 |     vec1.push(88);
  |     ^^^^ cannot borrow as mutable

error: aborting due to previous error

For more information about this error, try `rustc --explain E0596`.
error: could not compile `playground`.

To learn more, run the command again with --verbose.
```

After properly declaring the variable ***vec1*** as mutable, its value can be changed in its scope. Following is a screenshot for the corrected program:

```rust
pub fn main() {
    let vec0 = Vec::new();

    let mut vec1 = fill_vec(vec0);

    println!("{} has length {} content `{:?}`", "vec1", vec1.len(), vec1);

    vec1.push(88);

    println!("{} has length {} content `{:?}`", "vec1", vec1.len(), vec1);

}

fn fill_vec(vec: Vec<i32>) -> Vec<i32> {
    let mut vec = vec;

    vec.push(22);
    vec.push(44);
    vec.push(66);

```

Execution

Standarc

```
   Compiling playground v0.0.1 (/playground)
    Finished dev [unoptimized + debuginfo] target(s) in 1.60s
     Running `target/debug/playground`
```

Standard

```
vec1 has length 3 content `[22, 44, 66]`
vec1 has length 4 content `[22, 44, 66, 88]`
```

2. “move_semantics2.rs”:

The scope of a variable is worth focusing on when security is concerned. If a variable can be accessed even after its scope has ended, it can have serious security implications like modification by an attacker and accessing after modification can change the meaning of the instructions as well. In the program below, the program is trying to access variable ***vec0*** after its scope has ended. Once it is passed as an argument to the function, the parameter assumes the value of ***vec0*** ending its scope:

```
pub fn main() {
    let vec0 = Vec::new();

    let mut vec1 = fill_vec(vec0);
    println!("{} has length {} content `{:?}`", "vec0", vec0.len(), vec0);

    vec1.push(88);

    println!("{} has length {} content `{:?}`", "vec1", vec1.len(), vec1);

}

fn fill_vec(vec: Vec<i32>) -> Vec<i32> {
    let mut vec = vec;

    vec.push(22);
    vec.push(44);
    vec.push(66);

    vec
```

Execution

Standard Error

```
   Compiling playground v0.0.1 (/playground)
error[E0382]: borrow of moved value: `vec0`
 --> src/main.rs:5:57
  |
2 |     let vec0 = Vec::new();
  |         ---- move occurs because `vec0` has type `std::vec::Vec<i32>`, which does not implement the `Copy` trait
3 |
4 |     let mut vec1 = fill_vec(vec0);
  |                             ---- value moved here
5 |     println!("{} has length {} content `{:?}`", "vec0", vec0.len(), vec0);
  |                                                         ^^^^ value borrowed here after move

error: aborting due to previous error

For more information about this error, try `rustc --explain E0382`.
error: could not compile `playground`.
```

We declared another variable, cloned ***vec0***, passed this new variable to the function to avoid moving ***vec0*** and allowing the access of the variable. Following is the corrected program:

```
pub fn main() {
    let vec0 = Vec::new();
    let vec2 = vec0.clone();
    let mut vec1 = fill_vec(vec2);

    // Do not change the following line!
    println!("{} has length {} content `{:?}`", "vec0", vec0.len(), vec0);

    vec1.push(88);

    println!("{} has length {} content `{:?}`", "vec1", vec1.len(), vec1);

}

fn fill_vec(vec: Vec<i32>) -> Vec<i32> {
    let mut vec = vec;

    vec.push(22);
    vec.push(44);
    vec.push(66);
```

Execution

Standard

```
   Compiling playground v0.0.1 (/playground)
    Finished dev [unoptimized + debuginfo] target(s) in 0.87s
     Running `target/debug/playground`
```

Standard C

```
vec0 has length 0 content `[]`
vec1 has length 4 content `[22, 44, 66, 88]`
```

3. “move_semantics3.rs”:

In the last program, function ***fill_vec()*** declared another mutable variable that was mutated later in the function. But in this program, no variable is declared and the parameter itself is mutated in the function later. But by default a variable is immutable and cannot be mutated during its lifetime unless declared mutable. The security implication in this case is similar to what we discussed in the last problem. Following is a screenshot for the error:

```
pub fn main() {
    let vec0 = Vec::new();

    let mut vec1 = fill_vec(vec0);

    println!("{} has length {} content `{:?}`", "vec1", vec1.len(), vec1);

    vec1.push(88);

    println!("{} has length {} content `{:?}`", "vec1", vec1.len(), vec1);

}

fn fill_vec(vec: Vec<i32>) -> Vec<i32> {
    vec.push(22);
    vec.push(44);
    vec.push(66);

    vec
}
```

Execution

Standard

```
   Compiling playground v0.0.1 (/playground)
error[E0596]: cannot borrow `vec` as mutable, as it is not declared as mutable
  --> src/main.rs:15:5
   |
14 | fn fill_vec(vec: Vec<i32>) -> Vec<i32> {
   |             --- help: consider changing this to be mutable: `mut vec`
15 |     vec.push(22);
   |     ^^^ cannot borrow as mutable

error[E0596]: cannot borrow `vec` as mutable, as it is not declared as mutable
  --> src/main.rs:16:5
   |
14 | fn fill_vec(vec: Vec<i32>) -> Vec<i32> {
   |             --- help: consider changing this to be mutable: `mut vec`
15 |     vec.push(22);
16 |     vec.push(44);
```

After changing the function definition to make ***vec*** mutable, the value of ***vec*** can be mutated further in the function allowing successful compilation of the program. Following is a screenshot of the corrected program:

```rust
pub fn main() {
    let vec0 = Vec::new();

    let mut vec1 = fill_vec(vec0);

    println!("{} has length {} content `{:?}`", "vec1", vec1.len(), vec1);

    vec1.push(88);

    println!("{} has length {} content `{:?}`", "vec1", vec1.len(), vec1);

}

fn fill_vec(mut vec: Vec<i32>) -> Vec<i32> {
    vec.push(22);
    vec.push(44);
    vec.push(66);

    vec
}
```

Execution

Standar

```
   Compiling playground v0.0.1 (/playground)
    Finished dev [unoptimized + debuginfo] target(s) in 1.00s
     Running `target/debug/playground`
```

Standarc

```
vec1 has length 3 content `[22, 44, 66]`
vec1 has length 4 content `[22, 44, 66, 88]`
```

4. “move_semantics4.rs”:

We were asked to refactor the following program such that vector is created inside ***fill_vec()*** function instead of ***main()*** function. Following is a screenshot of the program before modification:

```
// Refactor this code so that instead of having `vec0` and creating the vector
// in `fn main`, we instead create it within `fn fill_vec` and transfer the
// freshly created vector from fill_vec to its caller. Scroll for hints!

pub fn main() {
    let vec0 = Vec::new();

    let mut vec1 = fill_vec(vec0);

    println!("{} has length {} content `{:?}`", "vec1", vec1.len(), vec1);

    vec1.push(88);

    println!("{} has length {} content `{:?}`", "vec1", vec1.len(), vec1);

}

fn fill_vec(vec: Vec<i32>) -> Vec<i32> {
    let mut vec = vec;

```

Execution

Standard Er

```
   Compiling playground v0.0.1 (/playground)
    Finished dev [unoptimized + debuginfo] target(s) in 0.59s
     Running `target/debug/playground`
```

Standard Out

```
vec1 has length 3 content `[22, 44, 66]`
vec1 has length 4 content `[22, 44, 66, 88]`
```

Instead of declaring a variable in ***main()*** function which is never used again after it is passed as parameter to the function ***fill_vec()***, we can declare and initialize a Vector type variable inside ***fill_vec()*** and return desired vector to ***vec1***. Following is the modified version of the program:

```rust
pub fn main() {
    //let vec0 = Vec::new();

    let mut vec1 = fill_vec();

    println!("{} has length {} content `{:?}`", "vec1", vec1.len(), vec1);

    vec1.push(88);

    println!("{} has length {} content `{:?}`", "vec1", vec1.len(), vec1);

}

fn fill_vec() -> Vec<i32> {
    let mut vec = Vec::new();

    vec.push(22);
    vec.push(44);
    vec.push(66);

```

Execution

Standard

```
   Compiling playground v0.0.1 (/playground)
    Finished dev [unoptimized + debuginfo] target(s) in 0.97s
     Running `target/debug/playground`
```

Standard

```
vec1 has length 3 content `[22, 44, 66]`
vec1 has length 4 content `[22, 44, 66, 88]`
```

Threads :

1. **“threads1.rs”:**

   In the program below, the threads are being tried to be accessed without concentrating much on concurrency. If threads can be accessed and modified concurrently, it can have security implications and leave the system in an unstable state. To avoid this, the access of threads needs locking it to tell that thread is in use and cannot be used while it is being accessed by another component. Following is a screenshot of the error thrown while we tried to run the program:

```rust
pub fn main() {
    //let vec0 = Vec::new();

    let mut vec1 = fill_vec();

    println!("{} has length {} content `{:?}`", "vec1", vec1.len(), vec1);

    vec1.push(88);

    println!("{} has length {} content `{:?}`", "vec1", vec1.len(), vec1);

}

fn fill_vec() -> Vec<i32> {
    let mut vec = Vec::new();

    vec.push(22);
    vec.push(44);
    vec.push(66);

```

Execution

Standard

```
   Compiling playground v0.0.1 (/playground)
    Finished dev [unoptimized + debuginfo] target(s) in 0.97s
     Running `target/debug/playground`
```

Standard

```
vec1 has length 3 content `[22, 44, 66]`
vec1 has length 4 content `[22, 44, 66, 88]`
```

To rectify the program, we first locked the thread, unwrapped it to access the thread. Following is a screenshot for the same:

```rust
use std::sync::{Mutex, Arc};
use std::thread;
use std::time::Duration;

struct JobStatus {
    jobs_completed: u32,
}

fn main() {
    let status = Arc::new(Mutex::new(JobStatus { jobs_completed: 0 }));
    let status_shared = status.clone();
    thread::spawn(move || {
        for _ in 0..10 {
            thread::sleep(Duration::from_millis(250));
            status_shared.lock().unwrap().jobs_completed += 1;
        }
    });
    while status.lock().unwrap().jobs_completed < 10 {
        println!("waiting... ");
        thread::sleep(Duration::from_millis(500));
```

Execution

Standard Err

```
   Compiling playground v0.0.1 (/playground)
    Finished dev [unoptimized + debuginfo] target(s) in 0.94s
     Running `target/debug/playground`
```

Standard Out

```
waiting...
waiting...
waiting...
waiting...
waiting...
waiting...
```

References:

[1] https://doc.rust-lang.org/book/second-edition/ch08-02-strings.html

[2] https://doc.rust-lang.org/book/second-edition/ch16-01-threads.html

[3] https://doc.rust-lang.org/book/second-edition/ch03-02-data-types.html

[4] https://doc.rust-lang.org/book/second-edition/ch03-01-variables-and-mutability.html

[5] https://doc.rust-lang.org/book/second-edition/ch03-03-how-functions-work.html

[6] https://doc.rust-lang.org/book/second-edition/ch16-01-threads.html