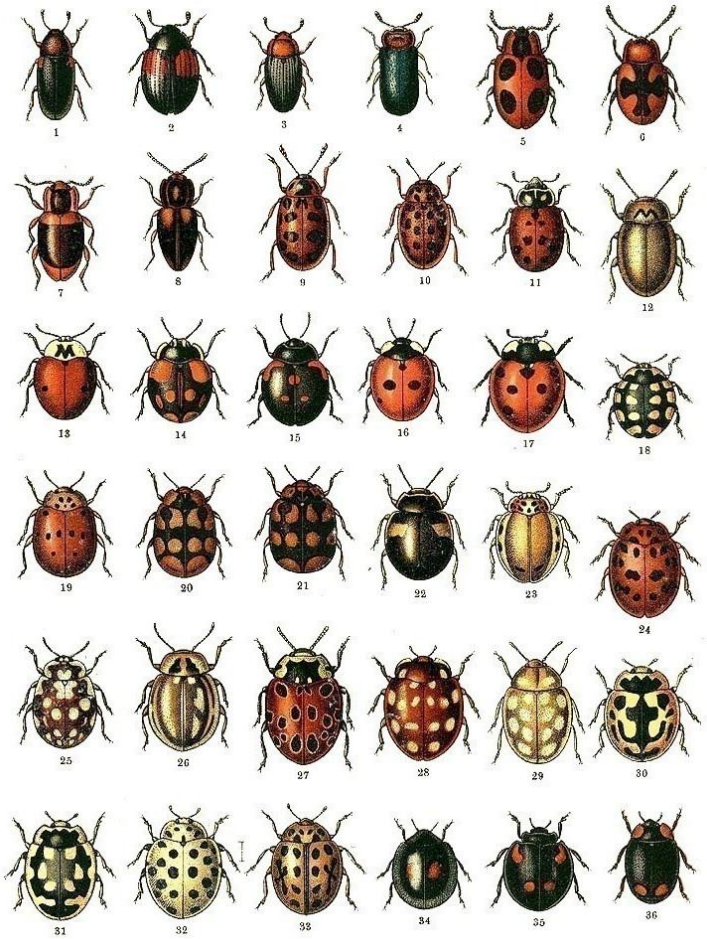

Seeing the bigger picture
Or, “Can you see the bug?”
Andrzej Krzemieński

My hobby

- Cataloguing bugs
- Identifying patterns
- Drawing conclusions



Agenda

- Authentication example
  - Find the bug
  - Find the root cause
  - Take precautions

Dealing with bugs

- Taking actions in run-time?
  - no good action exists
- Fixing bugs?
  - may be too late
- Not having bugs!
  - possible?

Authentication example

Find the bug

```
bool does_user_exist(string user_name)
{
  string query =
   "select count(*) from USERS where NAME = \'"
   + user_name + "\';";
  return DB::run_sql<int>(query) > 0;
}
```

- Can you see any bug?

Find the bug

```
bool does_user_exist(string user_name)
{
  string query =
   "select count(*) from USERS where NAME = \'"
   + user_name + "\';";
  return DB::run_sql<int>(query) > 0;
}
```

```
bool autheticate()
{
  string user_name = UI::read_user_input();
  return does_user_exist(user_name);
}
```

This is our caller

- Can you see any bug now?

Find the bug

User input:

```
JOHN'; delete from USERS where 'a' = 'a
```

Final SQL query:

```
select count(*) from USERS where NAME = 'JOHN';
delete from USERS where 'a' = 'a';
```

SQL Injection!

```
bool does_user_exist(string user_name)
{
  string query =
   "select count(*) from USERS where NAME = \'"
   + user_name + "\';";
  return DB::run_sql<int>(query) > 0;
}
```

Find the bug

Is a hacker attack a bug?

- User is allowed to enter any input – no bug here.
- Allowing unprocessed user input too deep into the program is a bug.
- User authentication module that wipes out a DB table – definitely a bug.
  - You do not need a “user story” to claim that.

Find the bug

„Let's deliver what we have, and fix bugs in the next release."

- This is not life-critical software.
- Still, the cost can be more than just customer complaints.
- There may be no next release.

Find the bug

But it is impossible to find it if you do not know it is there

- You can’t:
  - You wrote it, you are tired of it
  - Your motivation: proving that you are no good
- Ask your colleague:
  - His motivation: proving that he is better
  - He is new to the problem
- Help your colleague find it:
  - Make the code clean
  - Put comments, hints, assumptions

Find the root cause

- How did this happen?
- Where exactly is the bug located?
- Where to fix it?

```
bool does_user_exist(string user_name)
{
  string query =
   "select count(*) from USERS where NAME = \'"
   + user_name + "\';";
  return DB::run_sql<int>(query) > 0;
}
```

```
bool autheticate()
{
  string user_name = UI::read_user_input();
  return does_user_exist(user_name);
}
```

Find the root cause

`authenticate()`

- Should it have a check?
  - For what?
  - It doesn't know what the callee is doing.
  - It need not use SQL.

```
bool autheticate()
{
  string user_name = UI::read_user_input();
  return does_user_exist(user_name);
}
```

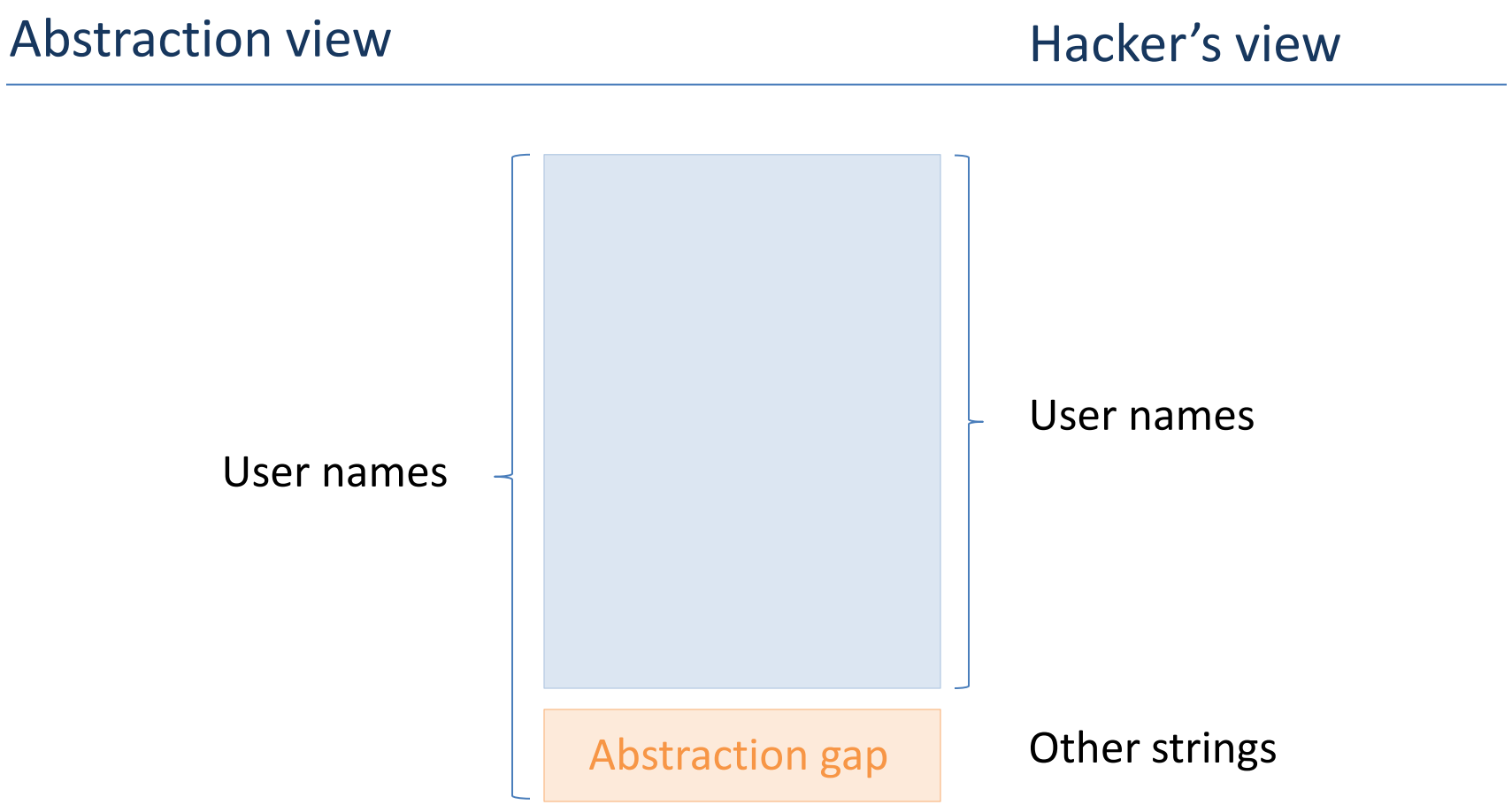
Find the root cause

`does_user_exist()`

```
bool does_user_exist(string user_name)
{
  string query =
   "select count(*) from USERS where NAME = \'"
   + user_name + "\';";
  return DB::run_sql<int>(query) > 0;
}
```

- Input is not “user name”
  - it is “any string”
- We are not querying for name in table USERS
  - We are concatenating strings
- The abstraction (“user name”) is for the callers
  - We must think lower-level, about “bits”
- Our abstraction works only within certain limit

Find the root cause



Find the root cause

Abstraction view

- Passing user names
- Querying for user name
- Abstraction, abstraction, …

- A tool for authenticating users

Hacker's view

- Passing any char sequence
- Concatenating strings
- Gap, gap, gap, …

- A tool for executing arbitrary SQL commands

Find the root cause

`does_user_exist()`

- Only a subset of values of `user_name` will work
- We have a *precondition*!
  - We failed to notice it

```
bool does_user_exist(string user_name)
{
  string query =
   "select count(*) from USERS where NAME = \'"
   + user_name + "\';";
  return DB::run_sql<int>(query) > 0;
}
```

Take precautions

First, name the gap (the precondition).

```
bool is_identifier(string s);
```

Second, communicate it to the callers.

- But why not check it ourselves?

Take precautions

No good recovery action exists here:

```
bool does_user_exist(string user_name) {
  if (!is_identifier(user_name)) {
    // do what?
  }
  // ...
}
```

Throw?

If caller can catch it, he may as well check the precondition.

Take precautions

The caller may know what to do:

```
bool autheticate() {
   string name = UI::read_user_input();
   while (!is_identifier(name))
      name = UI::read_user_input("try again");

   return does_user_exist(name);
}
```

At this level it is not an „error”.

Take precautions

So, we provide a pair:

```
bool is_identifier(string s);
bool does_user_exist(string user_name);
```

How do we communicate it to the callers?

- If you are determined you will do it. This way or the other.
- There is no ideal way.

Take precautions

Comment the declaration, possibly with custom syntax:

```
/*@ requires is_identifier(user_name); */
bool does_user_exist(string user_name);
```

(ACSL syntax)

```
bool does_user_exist(string user_name)
/* precondition{is_identifier(user_name)} */;
```

(My invented syntax)

Take precautions

Contract++ library:

```
CONTRACT_FUNCTION(
  bool (does_user_exist) (string user_name)
  precondition(is_identifier(user_name))
);
```

- Preconditions are syntax-checked
- Inserts run-time checks
  - Only for double-checks: terminate on failure
- Lots of macro and meta-programming magic!

Take precautions

Assertions:

```
bool does_user_exist(string user_name) {
  assert (is_identifier(user_name));
  // ...
}
```

- Syntax-checked comment
- Declaration of intent/assumptions
- Not visible in function declaration
- Use custom, improved assertion tool

Take precautions

Employ the type system:

```
bool does_user_exist(Identifier user_name);

bool autheticate()
{
  string user_name = UI::read_user_input();
  return does_user_exist(user_name); // error
}
```

Take precautions

```
class Identifier
{
   string value;
public:
   explicit Identifier(string); // has precondition
   string const& get() const;
};
```

- A kind of “opaque typedef”
- Only two conversions
- Sole purpose: alter the type system

Take precautions

```
bool autheticate()
{
   string user_name = UI::read_user_input();
   return does_user_exist(Identifier(user_name));
}
```

- Bugs still possible, but…
- You cannot do it by mistake
- We can easily locate the culprit

Take precautions

```
bool autheticate()
{
  Identifier user_name = UI::read_identifier();
  return does_user_exist(user_name);
}
```

- Don’t use conversions: propagate the type
- It is a useful popular abstraction
  - The effort to add it will pay off
  - Opportunity for optimizations (identifiers are short)

Take precautions

Constraint types can help more:

```
bool has_file(string path, string owner);
bool has_file(Path path, Identifier owner);
```

```
has_file(owner, path);    // bad argument order
sort(vec.begin(), vec.end(), has_file); // ouch
```

- Bad order of arguments
- Accidental signature match

Example – wrap-up

- Find the bug:
  - See the bigger picture
  - Review and help review
- Find the root cause:
  - Abstractions are for users, we work with bits
  - Identify your preconditions
- Take precautions
  - Declare your contract
  - If possible, to C++ compiler

Exercise on perceptiveness

```
bool is_identifier(string s) {
  return all_of(s.begin(), s.end(), isalnum);
}
```

Can you see the bug?

Exercise on perceptiveness

```
bool is_identifier(string s) {
  return all_of(s.begin(), s.end(), isalnum);
}
```

- Empty string?
- String “2”?
- Underscore?
- What about regional characters, ą, ź, ż, ö?
- Different locale?
- UTF8 encoding?