Modern Type Systems and Security

- Make assumptions explicit
- Eliminate undefined behaviour

Causes of Security Vulnerabilities

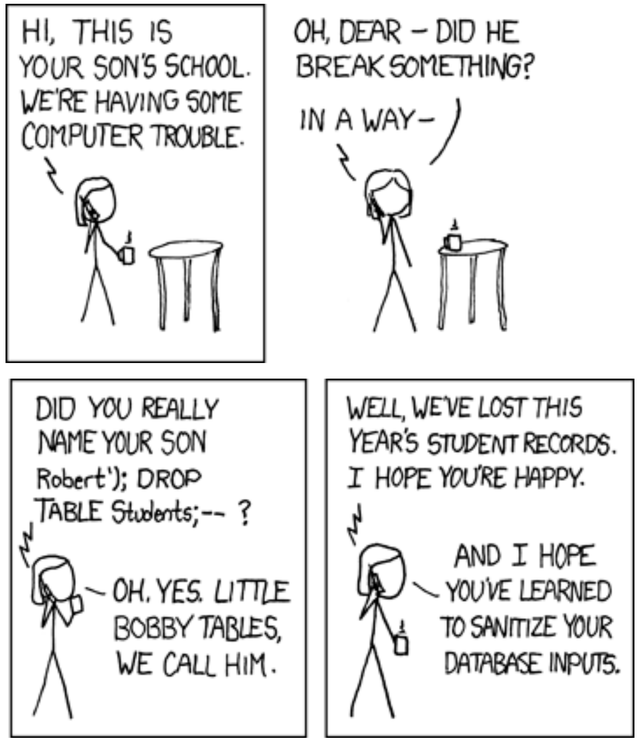
- Security vulnerabilities generally caused by persuading a program to do something that the programmer did not expect
  - Write past the end of a buffer
  - Treat user input as executable
  - Confuse a permission check
  - …
- Violate an assumption in the code

Causes of Security Vulnerabilities

- Strong typing makes assumptions explicit
  - Use explicit types rather than generic types
  - Define safe conversion functions
  - Use phantom types where necessary, to apply semantic tags to data

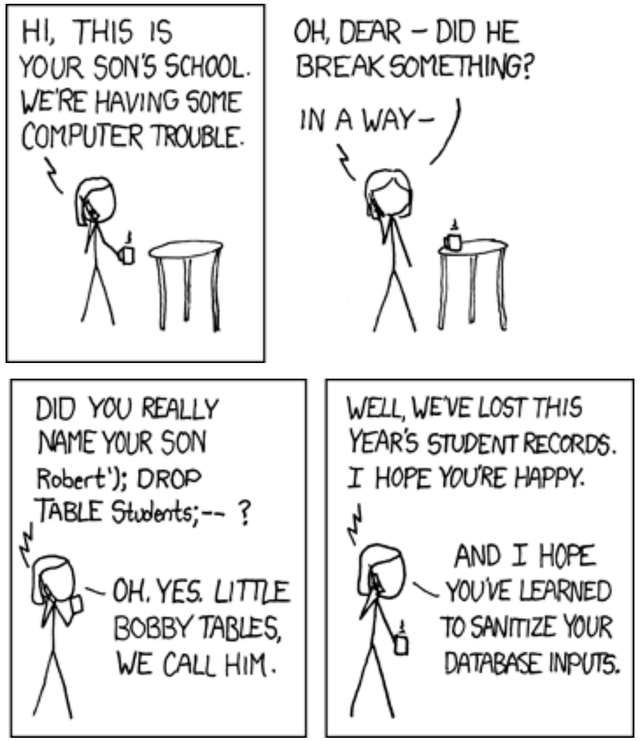
Prefer Explicit Types

- Vulnerabilities come from inconsistent data processing:
  - e.g., passing un-sanitised user entered data to a function that expects valid SQL
    - A **`StudentName`** and an **`SqlString`** are different; give them different types so compiler can catch inconsistent usage
    - Certain characters must be escaped before an arbitrary student name can be safely stored in an SQL database
- If **`String`** used everywhere, the programmer must manually check for consistency
  - Easy to make mistakes
  - If all the types are the same, the compiler can't help



Convert Data Carefully

- Explicit types require type conversions
  - Enforce security boundaries
  - Untrusted user input → escaped, safe, internal form; validate input before using it
  - Ensure only legal conversions occur



Use Phantom Types to Add Semantic Tags

- A phantom type parameter is one that doesn't show up at runtime, but is checked at compile time
  - In Rust, a **`struct`** with no fields has a type but is zero sized
  - Useful as type parameters to add semantic tags to data:

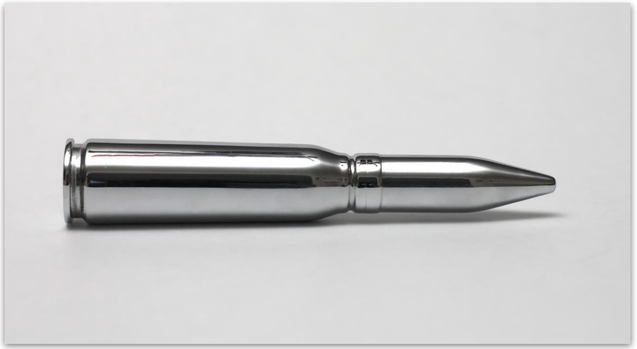
```
struct UserInput;  // As received from the user
struct Sanitised;  // After HTML codes have been escaped

fn sanitise_html(html : Html<UserInput>) -> Html<Sanitised> {
   …
}
```

  - Useful to represent states in a state machine

No Silver Bullet

- Memory safety and strong typing won’t eliminate security vulnerabilities
- But, used carefully, they eliminate certain classes of vulnerability, and make others less likely by making hidden assumptions visible



Liability and Ethics

ACM code of ethics and professional conduct:

1.2 Avoid harm.

In this document, "harm" means negative consequences, especially when those consequences are significant and unjust. Examples of harm include unjustified physical or mental injury, unjustified destruction or disclosure of information, and unjustified damage to property, reputation, and the environment. This list is not exhaustive.

Well-intended actions, including those that accomplish assigned duties, may lead to harm. When that harm is unintended, those responsible are obliged to undo or mitigate the harm as much as possible. Avoiding harm begins with careful consideration of potential impacts on all those affected by decisions. When harm is an intentional part of the system, those responsible are obligated to ensure that the harm is ethically justified. In either case, ensure that all harm is minimized.

To minimize the possibility of indirectly or unintentionally harming others, computing professionals should follow generally accepted best practices unless there is a compelling ethical reason to do otherwise. Additionally, the consequences of data aggregation and

https://www.acm.org/binaries/content/assets/about/acm-code-of-ethics-and-professional-conduct.pdf

Security vulnerabilities and software failures routinely cause harm – can you justify your professional practice?

Summary

- Memory safety
- Parsing and LangSec
- Modern Type Systems and Security
- Ethics and Liability