

model-based fuzzers \longleftrightarrow mutation-based fuzzers
modify good input slightly

main problems:

fuzzer configuration scheduling exploration vs. exploitation

fuzzer tuning rank bugs according to severity

safe programming languages

pointer = starting address of object in memory

memory safety

- 1. pointers are allocated only through the programming language
- 2. pointers only access memory that belongs to the program

program is memory-safe if all executions are memory-safe

language is memory-safe if all programs are memory-safe

spatial memory safety: no read/write outside of bounds

temporal memory safety:

- 1. no dangling pointers
- 2. no use-after-free

type-safety: prevents memory from being interpreted as different type of data

type systems: allows program/compiler to commit to range of values a variable can hold

inferred typing: automatic detection of typing

type checking can be done at compile-time or at run-time

There can be type hierarchies; i.e. one type is a subtype of another

These types are sometimes known as complex types

In this case, we can have languages where variable types are automatically re-interpreted

nominal typing: type system relies on explicit information

structural typing: type system relies on structure of type

} to compare types

web security

ofo: data unintentionally interpreted as input

Algorithm 1: input validation & representation

SQL injection

ephemeral states

session hijacking

XSRF cross-site request forgery

XSS cross-site scripting

protocol: //serveraddress/path? argument1 & argument2

ofo: https

response contains static context or dynamically generated context

state could be stored in session cookies

server-side, e.g. key-value store
client-side, identifies session

we want an ephemeral state to be re-recognized using information for long-term state

GET: ofo: for reading; generally does not modify long-term state

parameters in URL

POST: data input; changes data more ofo: than not

response contains cookies which client should store to maintain ephemeral state

database systems: ACID

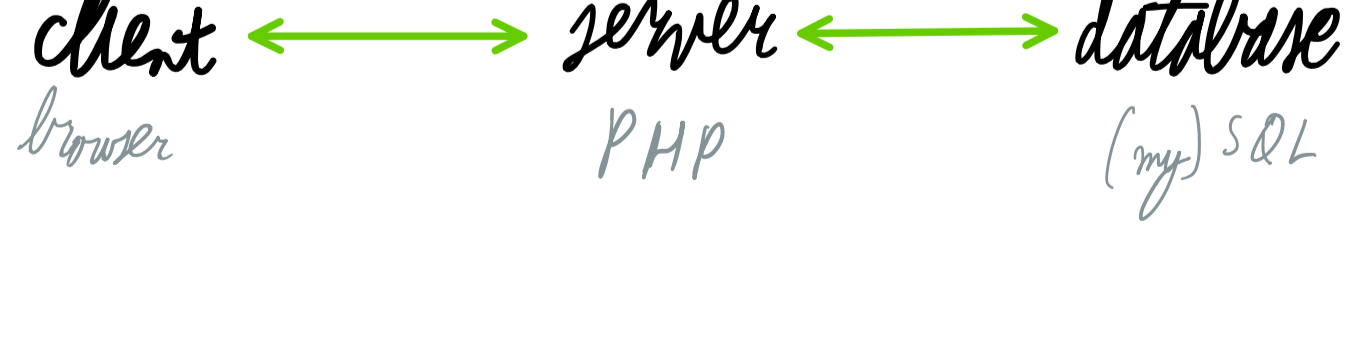
Atomic: statements are a single unit; they are fully executed or not at all

Consistent: all database states are valid; temporary invalid states are not written

Isolation: concurrently executed transactions behave as if executed sequentially

Durability: finished transactions remain persistent; i.e. are resistant to e.g. power failure

general structure:



SQL injection: data interpreted as code/commands

countermeasures: drop unexpected input forms

whitelisting

sanitize input

blacklisting

escaping

quoted statement \rightarrow fixes parts to be interpreted as either

command or data requested by ? symbols

no confusion

alternatives

restrictive access control policy especially for important tables

encrypt sensitive data