Lecture 9

Tuesday, 13 December 2022 10:44

model-based fuzzers ↔ mutation-based fuzzers

modify good input slightly

main problems:

fuzz configuration scheduling exploration vs. exploitation

fuzzer taming 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 them

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 } → to compare types

structural typing: type system relies on structure of type }

---

web security

often: data unintentionally interpreted as input kingdom 1: input validation & representation

SQL injection

ephemeral states

session hijacking

XSRF cross-site request forgery

XSS cross-site scripting

protocol://serveraddress/path ? argument1& argument2

↑ often https

response contains static content or dynamically generated content

state could be stored in session cookies ← client-side, identifies session

↑ server-side, e.g. key-value store

we want an ephemeral state (↑ cookies) to be re-recognized using information for long-term state

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

parameters in URL

POST : data input ; changes data more often 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:

client ↔ server ↔ database

browser PHP (my) SQL

SQL injection: data interpreted as code/commands

countermeasures: drop unexpected input forms

whitelisting

sanitise input

blacklisting

escaping

prepared statement → fixes parts to be interpreted as either command or data represented by ? symbols

↓ no confusion

alternatives

restrictive access control policy especially for important tables

encrypt sensitive data