Secure Programming Techniques

Margus Freudenthal, Mart Oruaas

Lecture 8, 2022-04-08
Programming Models
Introduction

- Programmers write program source code, usually in accordance to some kind of user story or use case documentation.
- It is easy to program the sequential “happy path” functionality of such use cases in any sane programming language.
- It is at least an order of magnitude more complex to understand how to program in such a way that all nonfunctional requirements (esp around security and performance) are fulfilled.
Program Design

- Motivation: OWASP A04:2021 – Insecure Design
- “Enumerating badness does not work”
- Any term in program source text tends to have relatively local effects on the overall execution
- “Design” is how those local effects compose to an overall program, doing something specific (*an informal definition*)
- Design is very much affected by the problem space, but also affected by the solution space – which machines are we using in solving the problem and how do we program them.
Understanding the Program

- Every properly designed programming language comes with a specification that tells at least something about two things:
  - The computational power of the language (most languages are Turing complete, but not all – take SQL as an example)
  - How a program written in the language behaves in terms of the underlying abstract machine.
Abstract Machine

- Abstract machine defines a hypothetical computer that is used to describe behaviour of programs
- Abstract machine gives basic building blocks for describing the behaviour: bootstrapping, memory model, type system etc
- Examples: C and Java language specifications
- Abstract machine behaviour is mapped to real machine behaviour by interpreters, compilers and virtual machines
- Those mappings can differ wildly, based on computer architecture.
An Example: Integer Overflows

- Most of the programming languages, esp the ones with C legacy, offer fixed-width integer types
- Values of those types over- and underflow
- E.g. an 8-bit signed integer with value 127 (01111111B) overflows to value -128 (10000000B), when 1 is added to it
- This can lead to all kinds of peculiar corner cases in program logic when not handled properly
- Some language offer infinite integers by default, though, it is important to know what kind of language are you using.
The Program Text Alone is Useless

- Every useful program is hooked to some kind of other programs: operating systems, frameworks, containers, virtual machines, etc.
- The meaning of the program (i.e. what the program does – including its vulnerabilities) stems from the combination of the language the program is written in and where the program hooks into
- We have already talked about supply chain problems, but there are more subtle issues
- One misguided myth: “In order to program in C, you need to understand assembler.”
How Can an External Factor Subvert the Programmer

- Java Memory Model gives relatively small and relaxed number of guarantees to program behavior
- Some JVM versions/implementations on some kind of hardware may give much stronger guarantees, and when the programmer relies on those guarantees, problems arise as soon as the environment changes [Shipilev, 2014] [Shipilev, 2016]
- Programmers need to stick to what language and system API-s guarantee
- It is not an easy task to figure out what the guarantees actually are! “Make it correct first and fast later.”
Why are C programs so Vulnerable?

- C language specification defines an abstract machine that dictates what C programs do.
- This AM is deliberately simple, because C programs need to be run on bare hardware as well (consider microcontrollers).
- C AM is pretty much a straightforward von Neumann machine.
- That simplicity leads to direct memory access to anything that the C program has access to, usually all the whole address space of the process. And when the “process” happens to be an OS kernel, all bets are off.
- Recent example: so-called DirtyPipe vulnerability, CVE-2022-0847.
Large Variability

- Modern world of computing is immensely heterogeneous, even if we talk about running programs written in the same language, like Java
- This was discussed in lecture 2
- Deployment models and platforms create their own abstract machines, but they are not always described with necessary stringency
- An example: the eventually-consistent nature of AWS S3 platform is really different from synchronous nature of any local file system, but the former is very often made to look like the latter on syntactic level. This has led to problems with data integrity and availability.
Mobile Applications

- Mobile platforms offer a significant number of unique services, which could and should be used. Usually, mobile application does not touch the native kernel too much.

- Mobile applications (iOS and Android native applications) have relatively complex lifecycles, with “crash” being pretty much a normal thing.

- One must take very good care when there are parts of application lifecycle that need to be preserved over the restart of the application.

- OWASP Top 10 Mobile Risks 2016

- M1: Improper Platform Usage
Web Browsers

- Web browsers started out as very simple tools for displaying hypertext documents.
- With introduction of programmability (JavaScript, later WebAssembly) and stateful behaviour (cookies etc), they were turned into very complex operating environments – browsers are pretty much considered operating systems these days.
- Keywords: Document Object Model (DOM), Origin, Same-Origin Policy, Cross-Origin Resource Sharing (CORS)
- Browser standards are constantly changing, it is necessary to keep oneself up to date!
- “The Tangled Web” [Zalewski, 2012]
Exotic but Interesting

- Intel SGX (Software Guard Extensions)
- ARM v9 CCA (Confidential Compute Architecture)
- Cybernetica’s Sharemind [Jaak Randmets, 2017]
Distributedness

- Distributed computing is everywhere these days.
- Every simple three-tier (web browser, web server, database engine) web is in reality a complex distributed application, with thin veneer of frameworks and standards hiding the complexity.
- Every single link between components of a distributed system is a source of vulnerabilities and there are times when frameworks hide those links in the most dangerous ways.
- Learn to tame the distributed systems [Shostack, 2014].
Concurrency and Parallelism

- Concurrency is the task of running and managing the multiple computations at the same time – meaning, they are interleaved in wallclock time.
- Parallelism is the task of running multiple computations simultaneously – meaning, they are actually running at the very same time on independent pieces of silicon.
- Slices of concurrent programs can be executed in parallel, but they do not need to be.
- If programs are written correctly, the nature of underlying hardware should not have any effect on the program semantics – the practice is usually different, though.
Concurrency is Still not Supported

Well

- In general, even in 2022, concurrency is poorly supported on programming language level
- The worst offender is probably Java, that had its concurrency model designed in the 90s and still exposes threads as primitives
- Go, Rust and Erlang are somewhat better with their corresponding language constructs
- The main problem with concurrency is state space explosion. Running two algorithms with 8 sub-states in parallel results in a state space with \( > 2^8 \) states. Correctness of such systems is close to impossible to verify by hand (assuming the algorithms are not completely independent).
Race conditions

- Race condition – correctness of the program depends on timing (race with an external entity)
  - Process scheduling can switch context any time, for any amount of time
  - Network is asynchronous by nature
Techniques of Taming Concurrency

- Do not share data between different execution contexts (see the next slide)
- Follow the locking advice given by language/runtime/framework documentation
- Pay attention to time-of-check-time-of-use (TOCTOU) problems
  - A program checks a property of a resource
  - An attacker alters the contents of the resource
  - The program later performs an action based on the initial check
What is and is not “Sharing Data”

- Data sharing is essential
- What is generally problematic is concurrent access to some set of bits without appropriate control mechanisms in place
- Shared object references, memory buffers, database records, etc
- Data sharing between concurrent execution contexts must be explicit and done correctly – handover is the best alternative to sharing
- Programmer needs to understand data locking primitives and data visibility rules in any particular technological environment: mutexes, memory barriers, database transactions etc.
Observability

- One of the main tenets of security, including in the context of post-compromise investigation and learning, is **observability**
- Generally, every program must output something about its execution: statistical measurements, log records, system events etc
- This topic is very often overlooked, only solved as an afterthought and then usually given to some junior developer to solve
- Forensic analysis.
Thank you for your attention.
References I

A programming language for application development of secure
multi-party computation.

Java Memory Model Pragmatics (transcript).
https://shipilev.net/blog/2014/jmm-pragmatics/.

Close Encounters of The Java Memory Model Kind.
https://shipilev.net/blog/2016/
close-encounters-of-jmm-kind/.
References II

Wiley.

*The Tangled Web.*  
No Starch Press.