CS 6371/4301: Advanced Programming Languages

Dr. Kevin Hamlen

Spring 2024

Today's Agenda

- Course overview and logistics
- Course philosophy and motivation
  - What is an "advanced" programming language?
  - Type-safe vs. Unsafe languages
  - Functional vs. Imperative programming
- Introduction to OCaml
  - The OCaml interpreter and compiler
  - An OCaml demo

Course Overview

- How to design a new programming language
  - specifying language formal semantics
  - bad language design and the “software crisis”
  - “new” programming paradigms: functional & logic
  - how to formally prove program correctness
- Related courses
  - CS 4337: Organization of Programming Languages
  - CS 5349: Automata Theory
  - CS 6301: Language-based Security
  - CS 6353: Compiler Construction
  - CS 6367: Software Verification & Testing

Course Logistics

- Class Resources:
  - Course homepage: www.utdallas.edu/~hamlen/cs6371sp23.html
  - My homepage: www.utdallas.edu/~hamlen
  - Tentative office hours: 1 hr immediately after each class
  - Email: hamlen AT utdallas DOT edu
- Grading
  - Homework: 25%
  - In-class quizzes: 15%
  - Midterm exam: 25%
  - Final exam: 35%
- Homework
  - 9 assignments: 6 programming + 3 written
  - Homework must be turned in by **1:05pm** on the due date. Programming assignments submitted through eLearning; written assignments submitted in hardcopy at start of class.
  - Late homeworks NOT accepted!
- Modality: in-person (lectures recorded for later review)

Homework Policy

- Students MAY work together with other current students on homework
- **You MAY NOT consult homework solution sets from prior semesters (or collaborate with students who are consulting them).**
- **CITE ALL SOURCES**
  - includes web pages, books, other people, etc.
  - citation is required even if you don't copy the source word-for-word
  - there is nothing wrong with using someone else's ideas as long as you cite it
  - you will not lose any marks or credit as long as you cite
- Violating the above policies is PLAGIARISM (cheating).
- Cheating will typically result in automatic failure of this course and possible expulsion from the CS program.
- It is much better to leave a problem blank than to cheat!
  - Usually ~60% is a B and ~80% is an A.
  - However, cheating earns you an F. It's not worth it!

Quizzes

- in-class on specified homework due dates
- about 15-20 min. each
- approximately 1 quiz per unit, so about 8 total
  - lowest one dropped, so you can miss one without penalty
  - other misses only permitted in accordance with university policy (e.g., illness with doctor’s note, etc.)
- closed-book, closed-notes
- think of them as extensions to the homework
  - length/difficulty similar to one or two homework problems
  - To prepare, be sure you can solve problems like those seen on the most recent homework in about 15-20 minutes each and *without group help!*

Difficulty Level

- Warning: This is a tough course for some
  - “strange” math, brain-bending programming style, some PhD-level material
  - difficulty ranked high by past students
- No required text book
  - few approachable texts cover this advanced material
  - no large pools of sample problems exist to my knowledge
  - useful texts:
    - book by Glynn Winskel available from UTD library
    - online text and several online manuals linked from webpage
  - Warning: Some online web resources devoted to this material that you may randomly find are INCORRECT (e.g., certain Wikipedia pages). Rely only on *authoritative* sources.
- What you’ll get out of taking this course
  - excellent preparation for PhD APL qualifier exam
  - solid understanding of language design & semantics
  - modern issues in declarative vs. imperative languages
  - deep connections between abstract logic and programming

About me…

- PhD & Masters from Cornell University, B.S. in CS & Math from Carnegie Mellon University
- Research: Computer Security, PL, Compilers
- Industry/Government Experience: Microsoft Research; PI for Navy, Air Force, Army, DARPA, NSF, NSA, …
- Personal
  - Christian
  - married, three sons (one 11-year-old, and twin 8-year-olds)
- Programming habits
  - C/C++ (for low-level work)
  - assembly (malware reverse-engineering)
  - C#, Java (toy programs)
  - Prolog (search-based programs)
  - Gallina/Coq (high-assurance algorithm development)
  - OCaml, F#, Haskell (everything else)

Course Plan

- Running case-study: We will design and implement a new programming language
- Code an interpreter in OCaml
  - OCaml ("Objective Categorical Abstract Meta- Language") is an open-source variant of ML
  - Microsoft F# is OCaml for .NET (but not fully compatible with OCaml, so don't use it for homework)
  - Coq/Gallina is better (and harder) than OCaml (see me if you want to use it for homework)
  - Warning: OCaml has a STEEP learning curve!
  - Pre-homework: Install OCaml
    - Go to the course website and follow the instructions entitled "To Prepare for the Course…" by next time

What is an “Advanced” Programming Language?

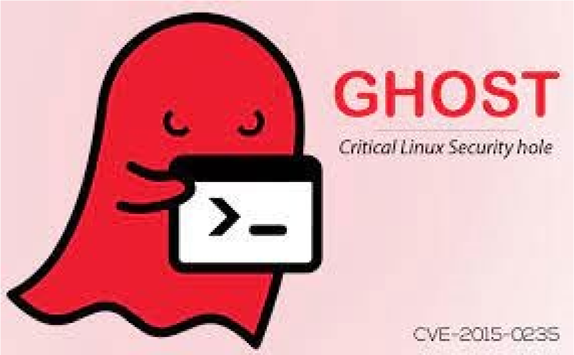
C/C++: Find the bug

```
int __nss_hostname_digits_dots( … ) {
    …
    size_needed = sizeof(*host_addr) + sizeof(*h_addr_ptrs) + strlen(name) + 1;
    *buffer = (char*)malloc(size_needed);
    … 35 lines of code …
    host_addr = (host_addr_t*)*buffer;
    h_addr_ptrs = (host_addr_list_t*) ((char*)host_addr + sizeof(*host_addr));
    h_alias_ptr = (char**)((char*)h_addr_ptrs + sizeof(*h_addr_ptrs));
    name = (char*)h_alias_ptr + sizeof(*h_alias_ptr);
    …
    if (isdigit(name[0])) {
        for (cp=name; ; ++cp) {
            if (*cp == '\0') {
                if (*--cp == '.') break;
                if ((af == AF_INET) ? inet_aton(name, host_addr) : inet_pton(af, name, host_addr))
                    result_buf->h_name = strcpy(hostname, name);
                goto done;
            }
            if (!isdigit(*cp) && *cp != '.') break;
        }
    }
```

C/C++: Find the bug

```
int __nss_hostname_digits_dots( … ) {
    …
    size_needed = sizeof(*host_addr) + sizeof(*h_addr_ptrs) + strlen(name) + 1;
    *buffer = (char*)malloc(size_needed);
    … 35 lines of code …
    host_addr = (host_addr_t*)*buffer;
    h_addr_ptrs = (host_addr_list_t*) ((char*)host_addr + sizeof(*host_addr));
    h_alias_ptr = (char**)((char*)h_addr_ptrs + sizeof(*h_addr_ptrs));
    name = (char*)h_alias_ptr + sizeof(*h_alias_ptr);
    …
    if (isdigit(name[0])) {
        for (cp=name; ; ++cp) {
            if (*cp == '\0') {
                if (*--cp == '.') break;
                if ((af == AF_INET) ? inet_aton(name, host_addr) : inet_pton(af, name, host_addr))
                    result_buf->h_name = strcpy(hostname, name);
                goto done;
            }
            if (!isdigit(*cp) && *cp != '.') break;
        }
    }
```

Impact of this C bug



- Discovered by Qualys researchers in 2015 during a routine code audit of the Gnu standard C libraries
  - affects nearly all Linux code that performs host lookups
- Initially classified as low-severity (rare crash)
- Qualys then demonstrated that they could use it gain complete remote control over nearly any Linux networking application.
- Eventual conclusion: Nearly all Linux systems were vulnerable to complete remote compromise for over a decade.

High-level Take-aways

- C/C++ code contains many “unsafe” features that invite disaster:
  - unconstrained pointer arithmetic
  - unstructured control-flows
  - unchecked datatype casting (programmer casts are blindly trusted)
  - in-lined assembly code
- About 25% of all highest severity bugs in history have been “buffer errors”.
- The world’s most mission-critical software (e.g., operating systems) consist of *hundreds of millions* lines of C code.
  - No human can comprehend, much less comprehensively debug/audit that.
- Most of the software crashes you experience are a direct result of the unsafe design of C/C++.

Java: A Type-safe, Imperative Language

- Find two bugs:

```
import java.io.*;
import java.util.*;

class Summation {
  public static void main(String[] args) {
    List list = new LinkedList();

    for (int i=0; i<args.length; ++i)
      list.add(args[i]);

    int sum = 0;
    while (!list.isEmpty())
      sum += ((Integer)list.remove(1)).intValue();

    System.out.println(sum);
  }
}
```

Java: A Type-safe, Imperative Language

- Find two bugs:

```
import java.io.*;
import java.util.*;

class Summation {
  public static void main(String[] args) {
    List list = new LinkedList();

    for (int i=0; i<args.length; ++i)
      list.add(args[i]);

    int sum = 0;
    while (!list.isEmpty())
      sum += ((Integer)list.remove(1)).intValue();

    System.out.println(sum);
  }
}
```

OutOfBounds Exception!

Cast Exception!

A Real-world Java Bug

```java
/**
 * Handles XML content
 */
public class XStreamHandler implements ContentTypeHandler {

    public String fromObject(Object obj, String resultCode, Writer out) throws IOException {
        if (obj != null) {
            XStream xstream = createXStream();
            xtream.toXML(obj, out);
        }
        return null;
    }

    public void toObject(Reader in, Object target) {
        XStream xstream = createXStream();
        xstream.fromXML(in, target);
    }

    protected XStream createXStream() {
        return new XStream();
    }

    …
```

A Real-world Java Bug

```java
/**
 * Handles XML content
 */
public class XStreamHandler implements ContentTypeHandler {

    public String fromObject(Object obj, String resultCode, Writer out) throws IOException {
        if (obj != null) {
            XStream xstream = createXStream();
            xtream.toXML(obj, out);
        }
        return null;
    }

    public void toObject(Reader in, Object target) {
        XStream xstream = createXStream();
        xstream.fromXML(in, target);
    }

    protected XStream createXStream() {
        return new XStream();
    }

    …
```

This is like a cast

“in” is untrusted!

Impact of this bug

- Discovered in Apache Struts library in 2017
  - Representational State Transfer (REST) plug-in
- Eventually identified as the root cause of the famous Equifax breach
  - Private financial data of over 150 million people stolen
  - One of the largest cybercrimes in history
  - Cost Equifax at least $650 million in fines (plus reputation loss, private settlements, etc.)

Problems with Java

- Every Java cast operation is potentially unsafe
  - Some casts are non-obvious (example: deserialization)
  - Even the obvious casts are so pervasive that they form a huge attack surface
- Some typecasting issues can be solved with Generics, but not all (e.g., list emptiness check)
- Problems:
  - Many forms of unsafe dynamic code-loading
  - Massive runtime library, whose foundations are mostly written in C
  - Inexpressive type system → code duplication → inconsistencies → bugs

Goals of Functional Languages

- In an “Advanced” Programming Language:
  - The compiler should tell you about typing errors in advance (not at runtime!)
  - The language structure should make it difficult to write programs that might crash (no unsafe casts!)
  - 80% of your time should be spent getting the program to compile, and only 20% on debugging
  - should be tractable to create a formal, machine-checkable proof of correctness for mission-critical core routines, or even full production-level apps

In OCaml…

- You almost never need to cast anything
  - The compiler figures out all the types for you
  - If there's a type-mismatch, the compiler warns you
- OCaml is fast
  - Somewhere between C (fastest) and Java (slow)
  - Hard to measure precisely. (So-called "language benchmarks" typically call underlying math libraries that aren't even implemented in the languages being tested!)
- Functions are "first-class":
  - you can pass them around as values, assign them to variables, …
  - you can SAFELY build them at runtime
- But: The syntax and coding style is very weird if you've only ever programmed in imperative languages!

OCaml: Getting Started

- OCaml programs are text files (*.ml)
  - Write them using any text editor (e.g., Notepad)
  - Unix: Emacs has syntax highlighting for ML/OCaml
  - Windows: I use Vim (www.vim.org)
- Installing OCaml (see course website)
  - Unix: pre-installed on the department Unix machines
  - Windows: Self-installers for native x86 and for Cygwin
- Two ways to use OCaml:
  - The OCaml compiler: ocamlc (compile *.ml to binary)
  - OCaml in interactive mode (use OCaml like a calculator)
  - Demo…