Formulario para la presentación de Cursos de Posgrado/Doctorado - Res. CD2819/18 - ANEXO 1

Información académica

Año de presentación (*)
2019

FACULTAD DE CIENCIAS EXACTAS Y NATURALES
FOLIO

1-a-

Departamento docente que inicia el tramite:
Departamento de Computación
Nombre del curso:
Tópicos de Compilación y Optimización de código
Nombre, Cargo y Título del docente responsable:
Juan Manuel Martínez Caamaño, Profesor Invitado, Doctor en Cs. Computación
En caso de dictarse en paralelo con una materia de grado, nombre de la misma:
Compilación y Optimización de código usando LLVM
Nombre y Título de los docentes que colaboran con el dictado del curso (*) (*):
--
Fecha propuesta para el primer dictado luego de la aprobación:
Noviembre de 2019

Duración:

Duración total en horas	30
Duración en semanas	2

Distribución carga horaria:

Número de horas de clases teóricas	15
Número de horas de clases de problemas	--
Número de horas de trabajos de laboratorio	15
Número de horas de trabajo de campo	--
Número de horas de seminarios	--

Forma de evaluación:
Un trabajo práctico final individual.
Lugar propuesto para el dictado (departamento, laboratorio, campo, etc.):
Laboratorio de computación

Puntaje propuesto para la carrera de doctorado:	2 puntos

Número de alumnos:	Mínimo: 5	Máximo: 40

Audiencia a quien está dirigido el curso:
Estudiantes de doctorado de la licenciatura en Cs. de la Computación.

Necesidades materiales del curso:
Laboratorio con proyector, pantalla y pizarrón Una computadora cada dos alumnos.

1-b-

Programa analítico del curso con Bibliografía (puede adjuntarse en hojas separadas):

CONTENIDO

Los compiladores modernos se organizan en 3 etapas: el frontend, encargado de leer el código fuente y asegurar que este sea válido; middle-end, donde la mayor parte de las transformaciones de código ocurren; y backend, que se encarga de emitir código ensamblador. En este curso nos vamos a concentrar en el middle-end.

Al final del curso los estudiantes deberían ser capaces de comprender la función del middle-end de un compilador, decisiones en su diseño, y poder programar análisis y transformaciones de código: análisis de dataflow, instrumentado, optimización, ofuscación, etc.

Para esto proponemos de utilizar el framework de compilación llamado LLVM (Low Level Virtual Machine). LLVM es un framework de código abierto, utilizado tanto en la academia como en la industria. Este incluye la especificación del bitcode ejecutado por una máquina virtual, llamado LLVM-IR, y una gama de proyectos que se comunican entre ellos utilizando este bitcode. Esto incluye frontends para diversos lenguajes (C/C++, Fortran, Swift, ...), frameworks de ejecución simbólica (KLEE), herramientas de instrumentado de código (ASAN, TSAN, USAN), backends para diversas arquitecturas (X86 64, arm32/64, mips, etc.), al mismo tiempo que una serie de análisis y optimizaciones de código implementadas sobre este bitcode.

PROGRAMA

El curso consiste en un total de 6 clases. Durante cada clase se alternan contenidos teóricos y la puesta en práctica de los mismos.

Clase 1 - Introducción

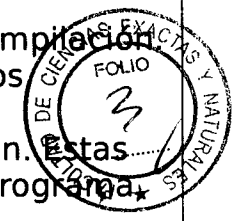
El objetivo de la primera clase es de introducir a los estudiantes a un toolchain de compilación moderna, a la arquitectura en 3 niveles clásica de los compiladores y a ciertas variaciones de esta.

- Arquitectura en 3 niveles: De código fuente a AST —> De AST a representación intermedia —> de representación intermedia a código máquina
- El linker y el loader
- Link-Time-Optimization y Thin-Link-Time-Optimization
- Profile-Guided-Optimization
- Just-In-Time compilation

Clase 2 - Transformaciones locales

Introducción al desarrollo dentro de LLVM. Representación intermediaria basada en Static-Single-Assignment. Primeras ofuscaciones: Mixed-Boolean-Arithmetic, Opaque-Constants y Opaque-Predicates.

Clase 3 - Transformaciones nivel unidad de compilación
Esta clase aborda transformaciones sobre la totalidad de la unidad de compilación. Esto implica tener en cuenta el tipo de linking y visibilidad de los símbolos globales del programa.
Esto se complementa con técnicas de Anti-Debug utilizadas en ofuscación. Estas detectan la presencia de un debugger y alteran el comportamiento del programa.



Clase 4 - Ingeniería inversa - Static Disassembly
Durante esta clase estudiaremos el enfoque de la herramienta RetDec. Esta herramienta puede transformar un binario en LLVM-IR, para luego ser analizado. Este LLVM-IR puede ser instrumentado, re-compilado y re-ejecutado para recolectar información útil para la ingeniería inversa.

Clase 5 - Inteniería inversa - Dynamic Symbolic Execution
En esta clase se estudiará el enfoque seguido por herramientas como Triton. Esta herramienta realiza ejecución concólica para explorar todos los posibles caminos de un ejecutable y reconstruir expresiones simbólicas dependientes de la entrada del programa para cada instrucción. También se abordan los análisis de tainting y backwards slicing utilizados por esta herramienta.

Clase 6 - Ingeniería inversa - Contramedidas
Estudiaremos las limitaciones de herramientas como RetDec y Triton, y como aprovecharlas para protegerse contra este tipo de herramientas.

BIBLIOGRAFÍA

- Compilers: Principles, Techniques, and Tools (2Nd Edition) - Alfred V. Aho and Monica S. Lam and Ravi Sethi and Jeffrey D. Ullman, 1986
- Lattner, C., & Adve, V. (2004). LLVM: A compilation framework for lifelong program analysis & transformation. In *Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization* (p. 75). IEEE. 2004
- Collberg, C. S., & Thomborson, C. (2002). Watermarking, tamper-proofing, and obfuscation-tools for software protection. *IEEE Transactions on Software Engineering*, *28*(8), 735-746, 2002.
- Guelton, S., Guinet, A., Brunet, P., Martinez, J. M., Dagnat, F., & Szlifierski, N. (2018). Combining Obfuscation and Optimizations in the Real World. In *2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM)* (pp. 24-33). IEEE, 2018.
- Code obfuscation against symbolic execution attacks - Sebastian Banescu and Christian S. Collberg and Vijay Ganesh and Zack Newsham and Alexander Pretschner, 2016
- Xu, H., Zhou, Y., Kang, Y., Tu, F., & Lyu, M. (2018). Manufacturing Resilient Bi-Opaque Predicates Against Symbolic Execution. In *48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (pp. 666-677). IEEE, 2018.
- Salwan, J., Bardin, S., & Potet, M. L. (2018). Symbolic deobfuscation: From virtualized code back to the original. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment* (pp. 372-392). Springer, Cham, 2018.
- RetDec: A Retargetable Machine-Code Decompiler - Avast Software
- Triton: A Dynamic Symbolic Execution Framework - Florent Saudel and Jonathan Salwan

1-c-

Actividades prácticas propuestas (puede adjuntarse en hojas separadas):
El curso tiene una orientación teórico-práctica y se desarrollará con trabajos de ejercitación en computadoras en el laboratorio, organizados en seis talleres (uno por cada clase): Taller 1: Familiarización con las distintas herramientas dentro de LLVM. Taller 2: Implementación de las ofuscaciones vistas, y de un dataflow analysis simple para hacer tainting. Taller 3: Implementación de la ofuscación Fault-Inject utilizada para anti-debugging. Taller 4: Instrumentado para detectar constantes ofuscadas. Taller 5: Uso de Triton Taller 6: Implementación de dos ofuscaciones, basadas en Opaque-Predicates y en Fault-Inject antes vistos, para protegerse de estas ofuscaciones.

(*) Todos los cursos tendrán una validez de 5 años
(*)(*) Las actualizaciones de los docentes colaboradores son informados por la Dirección departamental al inicio de cada dictado del curso

Firma Subcomisión Doctorado	Firma del docente responsable
ISABEL MÉNDEZ DÍAZ	AGUSTÍN GRAVANO (COORDINADOR PROGRAMA DE PROFESORES VISITANTES)

E-mail y teléfono del docente responsable

Agustin.gravano@gmail.com