Análisis estático de programas

Marcelo Arroyo, Francisco Bavera, Germán Regis *
Dpto de Computación - FCEFQyN
Universidad Nacional de Río Cuarto

Resumen

En el presente trabajo se describen las líneas de investigación en desarrollo por el grupo de investigación en análisis estático de propiedades de programas. Los integrantes del grupo atacan el problema del análisis de propiedades de programas mediante diferentes enfoques. Algunos trabajan a nivel de modelos mientras que otros lo hacen directamente a nivel de código fuente.

Se utilizan técnicas tanto de análisis estático liviano como pesados (basados en demostradores de teoremas o model-checking).

En una línea de trabajo se investiga en análisis estático de propiedades de programas a nivel de código fuente, usando diferentes técnicas como métodos livianos basados en interpretación abstracta y lógicas específicas como separation logic. Otra línea realiza análisis de propiedades de seguridad en programas en formatos de bajo nivel (bytecode) utilizando sistemas de tipos desarrollados a medida. La última línea de trabajo se basa en el análisis de propiedades de modelos de procesos de negocios usando métodos formales.

1. Introducción

El análisis de programas a partir de su código fuente o binario es una actividad que se ha desarrollado ampliamente en las ciencias de la computación. Los compiladores de lenguajes de programación son un ejemplo de analizadores de código fuente que además producen archivos ejecutables para plataformas de computación específicas.

Los compiladores certificantes generan, además del código ejecutable, pruebas de ciertas propiedades de los programas para que éstas puedan ser utilizadas para realizar verificaciones previas a su ejecución. Los analizadores de código fuente permiten detectar problemas sin necesidad de ejecutar los programas. Estos tipos de herramientas se están utilizando ampliamente para detectar problemas de seguridad.

En la primera sección se describe la línea de trabajo sobre análisis estático de programas a nivel de código fuente.

En la segunda sección se describen los avances en el área de código móvil seguro, específicamente desde el enfoque de garantías de propiedades de seguridad como la confidencialidad e integridad en *bytecode* en un entorno de código móvil.

Finalmente se describe la línea de verificación de propiedades de procesos de negocios.

*{marroyo,gregis,pancho}dc.exa.unrc.edu.ar

2. Análisis estático de propiedades de programas

En 1979 Bell Labs desarrolló *lint*, el cual se utilizaba para detectar código C sospechoso con el objetivo de poder detectar errores semánticos en programas, como uso de variables no inicializadas, condiciones invariables y operaciones cuyo resultado quedara fuera de rango.

Lint fue una de las primeras herramientas de análisis estático que trataba de detectar errores más allá de lo que hacía un compilador.

Actualmente la idea de análisis de código estático para la verificación de código fuente se ha ampliado y se han desarrollado diversas técnicas y herramientas, algunas de ellas aplicables a nivel industrial.

Algunas herramientas se utilizan para la verificación de propiedades de seguridad, como por ejemplo detección de posibles buffer overflows o inyección de código, mientras que otras tratan de detectar problemas de uso adecuado de componentes o detección de condiciones de carreras en código concurrente.

Las herramientas de análisis estático pueden clasificarse en:

- `Análisis estático liviano:` cuyo objetivo es analizar grandes cantidades de líneas de código en poco tiempo. Estas herramientas usan técnicas de sistemas de tipos extendidos, interpretación abstracta[2] y del uso de patrones estructurales. Tienen como desventaja la generación de muchos falsos positivos, por lo cual el usuario tiene que brindarles información mas precisa[1] para eliminarlos.

  Estas herramientas usan generalmente sistemas de reglas estructurales (o contextuales) para detectar problemas de mal uso de algunas construcciones,y anotaciones en forma de precondiciones, postcondiciones e invariantes en puntos específicos del programa. Esto permite que las herramientas sean *extensibles*, por lo que su exactitud depende de la información brindada por el usuario mediante anotaciones y/o la base de datos de reglas[3].

- `Análisis estático extendido:` Estas herramientas permiten la verificación de propiedades arbitrarias. Generalmente usan demostradores automáticos o semi-automáticos de teoremas (ej: SAT solving) o model checking y las herramientas requieren de un lenguaje de *anotaciones de programas*, como por ejemplo contratos Eiffel, JML (Java Modeling Language), SpecJ (Java) o Spec# (C#).

  Estas herramientas no son tan eficientes como las anteriores pero permiten demostrar propiedades mas precisas. Los avances producidos recientemente en SAT solving permiten analizar grandes volúmenes de código, por lo que su uso se está expandiendo en la industria[1].

La tabla 1 muestra algunas herramientas y sus características.

El grupo de investigación está desarrollando un framework abstracto para la implementación de técnicas de análisis estático como interpretación abstracta con reglas estructurales. Las propiedades a verificar serán implementadas por medio de una gramática de atributos.

[1] En forma de anotaciones en el mismo código fuente o mediante la incorporación de nuevas reglas a la herramienta.

Herramienta	Errores	Leng.	Técnicas usadas
SPLint	B. overflows, bounds, leaks, ...	C	Tipos, intraprocedural
Archer, BOON	idem	C	Ej. simbólica, flow-sens.
UNO	nil pointers, bounds	C	Model checking
FindBugs	bad downcasting, null refs, ...	Java	pattern matching
ESC/Java	props JML	Java	An. estático extendido

Figura 1: Herramientas de análisis estático para código fuente.

3. Confidencialidad e Integridad para *Bytecode*

Todos los sistemas operativos modernos incluyen algún mecanísmo de control de acceso para proteger los achivos de posibles lecturas o modificaciones por usuarios no autorizados. Sin embargo, controlar el acceso es una medida insuficiente para controlar la propagación de la información después que la misma a sido accedida por un programa comprometiendo la confidencialidad y/o integridad de los datos. Similarmente, la criptografía ofrece una fuerte garantía de preservar la confidencialidad pero el costo de realizar computaciones triviales con datos encriptados es muy costoso. Ninguno de estos dos enfoques provee una solución completa para proteger la confidencialidad.

Un enfoque complementario consiste en analizar y regular el flujo de la información (*information-flow*) [4] en el sistema para prevenir que se filtre datos privados a partes no autorizadas.

El análisis de confidencialidad e integridad para lenguajes de bajo nivel (assembly, bytecode) aún tiene un menor desarrollo que el alcanzado para lenguajes de alto nivel (Java, C). Esto se debe principalmente a la dificultad de razonar con programas no estructurados. Existen trabajos basados en sistemas de tipos que incluyen un subconjunto bastante extenso de Java bytecode pero no permiten reuso de variables locales (registros) ni de fields (atributos) de los objetos; otros trabajos permiten reuso de variables locales pero no contemplan objetos; además, estos trabajos son muy restrictivos con el uso de la pila de operandos.

Esto motivó el desarrollo de un sistema de tipos que soluciona las restricciones anteriores [6]. Este sistema, permite verificar que los programas Java Bytecode no violan la política de confidencialidad y/o integridad del usuario garantizando no-interferecia [5]. La propiedad de no interferencia establece que los datos públicos de salida de un programa no dependen de los datos privados de entrada.

No interferencia es una buena base de partida para generar herramientas que garanticen flujos de información segura, pero no es una propiedad que satisfagan la gran mayoría de los programas. Esto se debe a que muchos programas permiten revelar cierta información privada. Este problema implica la necesidad de contar con mecanísmos que permitan "liberar" (*declasificar*) información privada. Pero, es importante determinar que garantías de seguridad se pueden ofrecer ante tales mecanísmos.

Numerosos trabajos se direccionan en extender el análisis para que soporte declasificación de información [9]. Un enfoque interesante es *Robust declassification* [8]. *Robust declassification*, el cual garantiza que un atacante no puede deducir ninguna información adicional a la permitida por la política de seguridad ni tampoco puedo influir en los datos que serán desclasificados.

En nuestro grupo, se desarrolló el primer sistema de tipos que garantiza *robust declasification* para programas bytecode[7]. En este trabajo no sólo se probó la correción del sistema sino que también se determinó que los análisis de control de flujo sensitivos generan un nuevo problema en presencia de declasificación y que sistemas de tipos sensitivos deben ser revisados

si quieren ser extendidos con alguna noción de desclasificación (como *robust declasification*).

Dentro de la línea de investigación relacionada con confidencialidad e integridad, el grupo, pretende obtener el diseño e implementación de un sistema de tipos para bytecode multi-thread que garantice la política de seguridad e integridad especificada por los usuarios. La verificación de information-flow en presencia de concurrencia [9] no ha tenido mucho desarrolló y no se encuentra cerrado. Actualmente se encuentra un único trabajo que garantiza no-interferencia para aplicaciones bytecode concurrentes. Pero este enfoque plantea un uso específico del scheduler y una transformación de los programas compilados [10].

4. Análisis de propiedades de Procesos de Negocios

El objetivo general de este trabajo es la búsqueda de aplicaciones novedosas de técnicas formales de especificación, diseño y análisis estático a diferentes aspectos de las metodologías de desarrollo (generalmente informales) más ampliamente difundidas y utilizadas en la práctica. En particular, es nuestra intención centrarnos en técnicas formales de las llamadas "push-button", es decir, que ofrecen mecanismos de análisis potentes, que no requieren interacción fluida con el usuario.

El campo de aplicación específico es el modelado de procesos de negocios. Nuestro objetivo incluye además adaptar técnicas de análisis existentes para su utilización en este campo, y el estudio de mejoras a estas técnicas que pudieran surgir como consecuencia de su aplicación a dominios específicos.

El área de modelado de procesos de negocios ha tenido gran auge en los últimos años como consecuencia, en parte, de la necesidad y demanda de eficiencia de un mundo cada vez más competitivo. Es por ello que existen actualmente numerosos trabajos cuyo objetivo es brindar soporte para tal fin; algunos ejemplos de ellos son [15, 12].

En estos objetivos, la posibilidad de construir herramientas de software, o utilizar herramientas ya existentes, para asistir en la especificación y verificación de modelos de procesos de negocios se encuetra entre nuestras principales expectativas. Focalizamos nuestro interés fundamentalmente en métodos basados en formalismos relacionales y sistemas de transición de estados temporizados, intentando aprovechar el éxito de lenguajes y herramientas basados en este tipo de formalismos, tales como Alloy [13] y Uppaal [11].

Se está desarrollando un lenguaje para el modelado de procesos de negocios fundado en la idea para describir sistemas de información presentada en [14]. En ella, y a diferencia de la gran mayoría de los lenguajes formales o informales actuales para describir procesos de negocios, los productos (referencias a elementos pasivos del modelo) son elementos de primera clase dentro de las especificaciones de sistemas junto con los que referencian los procesos que los manipulan.

Conjuntamente con el desarrollo del mismo, se trabajó en la búsqueda de problemas que revelen la necesidad de contar con estas características para describir sistemas y, como fundamento de las ventajas de tratar con métodos formales, se investigó y definió una técnica para verificar (utilizando Model Checking) propiedades temporales de dichos sistemas [16]. Como herramienta de soporte se utilizó Uppaal[11].

Como trabajo futuro a corto plazo, se prevee el desarrollo de una herramienta para asistir al modelado de procesos de negocios utilizando el lenguaje definido. Otra actividad es la investigación y posible extención del lenguaje con el fin de cubrir diferentes instancias de procesos de negocios no soportados hasta el momento. Para ello se abordarán los diferentes patrones arquitecturales para el modelado de proceso de negocios y su implementación con

el lenguaje propuesto.

Referencias

[1] D. Letlefs, K. M. Laino, G. Nelson, J. Saxe. "Extended Static Checking". 1998. Compaq System Research Lab.

[2] P. Cousot. "Verification By Abstract Interpretation". Verification: Theory and Practice, pp 243-268. 2003.

[3] B. Chess, J. West. "Secure Programming with Static Analysis". Addison-Wesley. 2007.

[4] A. Sabelfeld and A. Myers. "Language-based information-flow security", 2003.

[5] Dennis Volpano, Geoffrey Smith, and Cynthia Irvine. "A sound type system for secure flow analysis". Journal of Computer Security , 4(3):167-187, 1996.

[6] Francisco Bavera, Eduardo Bonelli, "TypeBased Information Flow Analysis for Byteco-de Languages with Variable Object Field Policies". En Proceedings de el 23rd Annual ACM Symposium on Applied Computing (SAC'08). Marzo de 2008. Fortaleza, Brasil.

[7] F. Bavera, E. Bonelli, "Robust Declassification for Bytecode". CIBSI'09. Montevideo. Uruguay.

[8] Andrew C. Myers, Andrei Sabelfeld, and Steve Zdancewic. "Enforcing Robust Declassification and Qualified Robustness". Journal of Computer Security, 14(2):157-196, 2006.

[9] Andrei Sabelfeld, David Sands. "Dimensions and Principles of Declassification". CSFW 2005: 255-269.

[10] Gilles Barthe, Tamara Rezk, and Andrei Sabelfeld. "Security of Multithreaded Programs by Compilation". In Proceedings of the 10th European Symposium on Research in Computer Security (ESORICS), Dresden, Germany, September 24-26, 2007, LNCS, Springer-Verlag, September 2007.

[11] J. Bengtsson, K.G. Larsen, F. Larsson, P. Pettersson, and W. Yi. *UPPAAL- a tool suite for the automatic verification of real-time systems.* In Proceedings of Hybrid Systems III. LNCS 1066.pages 232-243. Spriger Verlag. 1996.

[12] H. Foster, et al. . *WS-Engineer: A Model-Based Approach to Engineering Web Service Compositions and Choreography.* Test and Analysis of Web Services, pp. 87-119, 2007.

[13] D. Jackson, Alloy: : A Lightweight Object Modelling Notation, ACM Trans. Softw. Eng. Methodol. 11(2): 256-290 (2002).

[14] M. Myers, A. Kaposi, *A First Systems Book: Technology and Management*, ISBN 978-1860944321, Imperial College Press, 2 edition, 2004.

[15] P. Y. H. Wong, J. Gibbons. *Property Specifications for Workflow Modelling*, IFM 2009, pp. 56-71, 2009.

[16] G. Regis, N. Aguirre, T. Maibaum, *Specifying and Verifying Business Process Using PPML.* ICFEM 2009, LNCS 5885, pp. 737-759, 2009.