Pruebas de Mutación de Caja Negra para APIs Web

Ana B. Sánchez, Alberto Martin-Lopez, Sergio Segura, and Antonio Ruiz-Cortés

Smart Computer Systems Research and Engineering Lab (SCORE)
Research Institute of Computer Engineering (I3US), Universidad de Sevilla, Spain
{anabsanchez,alberto.martin,sergiosegura,aruiz}@us.es

Resumen Las Interfaces de Programación de Aplicaciones (APIs) web desempeñan un papel clave en la integración de aplicaciones, por lo que validar su correcto funcionamiento resulta crucial. La mayoría de técnicas de generación de casos de prueba en este ámbito son de caja negra y con frecuencia se evalúan con APIs para las que no se dispone del código fuente. Por ello, no es posible emplear pruebas de mutación tradicionales, y no podemos cuantificar la efectividad de las pruebas si la API no expone ningún error. Este artículo propone un enfoque de caja negra para evaluar la capacidad de detección de errores de las suites de pruebas para APIs web: en lugar de crear variantes defectuosas del código del programa (pruebas de mutación tradicionales), se crean mutantes de las salidas del programa (respuestas HTTP). Los oráculos de pruebas se aplican sobre dichos mutantes y, a medida que las pruebas fallan, la cobertura de mutación aumenta. Experimentos preliminares sugieren que existe correlación entre la cobertura de mutación de caja negra y caja blanca.

Keywords: Pruebas de mutación · APIs Web · JSON

1. Introducción

Las APIs web permiten la interacción de sistemas software heterogéneos y distribuidos a través de Internet [5]. Por ello, validar el correcto funcionamiento de las APIs web es crucial. Un error en una API podría provocar un comportamiento inesperado en las muchas aplicaciones que dependen de ella. Recientemente, han surgido numerosas propuestas para probar APIs web de manera automática, casi siempre con enfoques de caja negra [2]. Para mostrar la efectividad de las técnicas es habitual que se evalúen con APIs industriales para las que no se dispone del código fuente como, por ejemplo, las APIs de YouTube o Paypal. Sin embargo, la efectividad de dichas técnicas solo puede evaluarse en términos de fallos reales detectados en la API. Si la API no expone ningún error, o muy pocos, la capacidad de detección de errores de la técnica evaluada es desconocida. A diferencia de las pruebas de caja blanca, donde la cobertura de mutación sirve para evaluar la capacidad de detección de errores de un banco de pruebas, las técnicas de caja negra solo pueden recurrir a cubrir tanta funcionalidad de la API como sea posible [3], con la esperanza de encontrar algún fallo.

Este trabajo en progreso propone un novedoso enfoque de *pruebas de mutación de caja negra* para APIs web. En lugar de crear variantes defectuosas del sistema (mutación tradicional), se crean variantes defectuosas de las *salidas* del sistema, en nuestro contexto, las respuestas HTTP devueltas por la API. Los oráculos de pruebas se aplican sobre dichos mutantes de las respuestas. A medida que las pruebas fallan, los mutantes mueren y la capacidad de detección de errores del banco de pruebas aumenta. Experimentos preliminares con una API *open source* sugieren que existe correlación entre la cobertura de mutación de caja negra y caja blanca, demostrando la utilidad de nuestra propuesta cuando no se dispone del código del sistema, algo habitual en las pruebas de APIs web.

2. Operadores de mutación

En esta sección presentamos nuevos operadores de mutación que se aplican a la respuesta HTTP de servicios web. Una respuesta HTTP está compuesta por un cuerpo (normalmente texto), un código de estado (número entero entre 100 y 599) y un conjunto de cabeceras (pares clave-valor). Clasificamos los operadores de mutación de acuerdo a estos tres elementos. Para el cuerpo, nos centramos en JSON como el formato estándar de intercambio de datos en APIs web [5].

El proceso de definición de los operadores constó de dos fases. Primero, estudiamos la literatura relacionada con las pruebas de mutación en servicios web [1,4]. Detectamos que no se habían propuesto operadores de mutación para objetos JSON hasta la fecha. Por ello, y con la intención de definir operadores de mutación realistas, estudiamos los sistemas de seguimiento de errores de las APIs de Spotify[1] y YouTube[2]. Extrajimos y analizamos un total de 300 errores, de los cuales seleccionamos 121 como adecuados para nuestro propósito. En segundo lugar, derivamos los operadores de mutación basándonos en la información obtenida en el paso anterior, y los agrupamos en tres categorías: operadores de mutación para cabeceras de respuesta HTTP, operadores sobre el código de estado HTTP, y operadores de mutación para el cuerpo de respuesta JSON.

2.1. Operadores de mutación para cabeceras de respuesta HTTP

Estos operadores mutan la estructura o propiedades de una cabecera de respuesta HTTP. Simulan problemas reales que pueden ocurrir en las cabeceras de respuesta y que pueden producir que el mensaje no llegue al destino o no se muestre correctamente. La Tabla 1 presenta los 6 operadores de mutación que proponemos. Por cada operador, se detallan su nombre y una breve descripción.

2.2. Operadores de mutación para códigos de estado HTTP

Estos operadores mutan el código de estado de una respuesta HTTP. Los códigos de estado indican si una petición se ha resuelto de manera satisfactoria. La

[1] `https://github.com/spotify/web-api/issues`
[2] `https://issuetracker.google.com/issues?q=issues`

Nombre	Descripción
CCT	Cambia el valor de la cabecera Content-Type.
ECT	Elimina la cabecera Content-Type.
CPC	Cambia el valor de la propiedad Charset de la cabecera Content-Type.
EPC	Elimina la propiedad Charset de la cabecera Content-Type.
CL	Cambia el valor de la cabecera Location.
EL	Elimina la cabecera Location.

Tabla 1. Operadores de mutación para cabeceras de respuesta HTTP.

tabla 2 presenta 4 operadores de mutación para el código de estado de respuesta HTTP. Por cada operador, se detalla el nombre y una breve descripción.

Nombre	Descripción
R5XX	Reemplaza el código de estado por un código de error de servidor en el rango 5XX*.
R200-409	Reemplaza el código de estado con un código en el rango 200-409$^\diamond$.
R2XX	Reemplaza el código de estado por un código de estado de éxito en el rango 2XX$^\diamond$).
R4XX	Reemplaza el código de estado con un código de estado de error en cliente en el rango 4XX $^\diamond$.

$\star$ Códigos de estado de error de servidor seleccionados: 500, 502, 503, 504.
$\diamond$ Códigos de estado seleccionados: 200, 201, 204, 400, 401, 403, 404, 409.

Tabla 2. Operadores de mutación para el código de estado de respuesta HTTP.

2.3. Operadores de mutación para objetos JSON

Estos operadores modifican el cuerpo de respuesta de un mensaje en formato JSON mediante la edición, eliminación o inserción de propiedades, objetos o arrays. Estas mutaciones pueden provocar errores en los datos que llegan al cliente: datos incompletos, información incorrecta, datos vacíos, etc. La tabla 3 muestra los 14 operadores propuestos en esta sección. Por cada operador, detallamos el nombre, descripción y contexto de aplicación de la mutación.

Nombre	Descripción	Contexto
AEA	Añade un elemento a un array.	*Arrays*
EEA	Elimina un elemento de un array.	*Arrays*
IEA	Intercambia un elemento por otro en un array.	*Arrays*
ANA	Asigna el valor nulo a un array.	*Arrays*
APO	Añade una propiedad a un objeto.	*Objetos*
EPO	Elimina una propiedad de un objeto.	*Objetos*
EPOO	Elimina una propiedad de tipo objeto de un objeto.	*Objetos*
CTP	Cambia el tipo de una propiedad.	*Propiedades*
IPB	Invierte el valor de una propiedad de tipo booleano.	*Propiedades*
ANP	Asigna el valor nulo a una propiedad.	*Propiedades*
RPN	Reemplaza el valor de una propiedad de tipo numérico con otro valor.	*Propiedades*
ACSP	Añade caracteres especiales a una propiedad de tipo cadena.	*Propiedades*
RCP	Reemplaza la cadena completa de una propiedad por otra cadena.	*Propiedades*
RECP	Reduce/Extiende la cadena de una propiedad a un longitud límite.	*Propiedades*

Tabla 3. Operadores de mutación para el cuerpo de respuesta en formato JSON.

3. Evaluación preliminar

La Tabla 4 muestra los resultados de la evaluación realizada con YouTube-Mock[3] (API REST de unas 3000 líneas de código). Se ha evaluado la propuesta con cuatro bancos de pruebas, ordenados por tamaño y exhaustividad, de acuerdo a los criterios de cobertura para APIs REST definidos en nuestro trabajo

[3] `https://github.com/opensourcingapis/YouTubeMock`

previo [3]. Los operadores de mutación de caja blanca utilizados son los habilitados por defecto en la herramienta PIT[4], los de caja negra son los descritos en la sección anterior. El número de mutantes en nuestra propuesta (sexta columna) está determinado por el número de casos de prueba, ya que por cada salida del programa se generan múltiples mutantes. El banco de pruebas TCL3 (menor tamaño y exhaustividad) consigue un 12 % de cobertura de caja blanca y un 0 % de caja negra, ya que los oráculos utilizados no consiguen detectar los cambios introducidos en las salidas del programa (respuestas HTTP mutadas). Por contra, el banco de pruebas TCL6 (mayor tamaño y exhaustividad) consigue un 57 % y 51 % de cobertura de caja blanca y negra, respectivamente, lo que sugiere que nuestra propuesta es efectiva para evaluar la capacidad de detección de errores cuando no se dispone del código fuente del programa.

Test suite	Casos de prueba	Caja blanca			Caja negra	
		Mutantes	Cobertura de código	Cobertura de mutación	Mutantes	Cobertura de mutación
TCL3	1	723	69 %	12 %	852	0 %
TCL4	6	723	81 %	14 %	2797	45 %
TCL5	18	723	86 %	32 %	6878	45 %
TCL6	19	723	87 %	57 %	7082	51 %

Tabla 4. Cobertura de caja blanca y caja negra en función del tamaño y exhaustividad del banco de pruebas.

4. Trabajo futuro

Nuestro trabajo futuro perseguirá una evaluación más exhaustiva y con más APIs web que nos permita confirmar la validez de la propuesta y estudiar más a fondo la correlación entre cobertura de mutación de caja negra y caja blanca.

5. Agradecimientos

Trabajo parcialmente financiado por la Comisión Europea (FEDER), la Junta de Andalucía bajo los proyectos APOLO (US-1264651) y EKIPMENT-PLUS (P18-FR-2895), el Gobierno de España bajo el proyecto HORATIO (RTI2018-101204-B-C21), financiado por: FEDER/Ministerio de Ciencia e Innovación – Agencia Estatal de Investigación y el Programa de becas FPU, concedido por Ministerio de Educación y Formación Profesional de España (FPU17/04077).

Referencias

1. de Melo, A.C., Silveira, P.: Improving data perturbation testing techniques for web services. Information Sciences **181**(3), 600–619 (2011)
2. Martin-Lopez, A., Segura, S., Ruiz-Cortés, A.: RESTest: Black-Box Constraint-Based Testing of RESTful Web APIs. In: ICSOC. pp. 459–475 (2020)
3. Martin-Lopez, A., Segura, S., Ruiz-Cortés, A.: Test Coverage Criteria for RESTful Web APIs. In: A-TEST. pp. 15–21 (2019)
4. Offutt, J., Xu, W.: Generating test cases for web services using data perturbation. SIGSOFT Softw. Eng. Notes **29**(5), 1–10 (2004)
5. Richardson, L., Amundsen, M., Ruby, S.: RESTful Web APIs. O'Reilly Media, Inc. (2013)

[4] `https://pitest.org/quickstart/mutators/`