ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS

UNIDAD DE TITULACIÓN

ANÁLISIS CORRELACIONAL ENTRE MÉTRICAS DE PROCESO Y DEFECTOS DE SEGURIDAD EN SCRIPTS DE INFRAESTRUCTURA COMO CÓDIGO (IaC)

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL GRADO DE MAGISTER EN SOFTWARE

CARLOS ANDRES CUEVA ANCHAPAXI
anndres.cueva@gmail.com

Director: Pamela Catherine Flores Naranjo, PhD
pamela.flores@epn.edu.ec

2020

APROBACIÓN DEL DIRECTOR

Como director del trabajo de titulación Análisis Correlacional entre Métricas de Proceso y Defectos de Seguridad en Scripts de Infraestructura como Código (IaC) desarrollado por Carlos Andrés Cueva Anchapaxi, estudiante del programa de maestría en Software, habiendo supervisado la realización de este trabajo y realizado las correcciones correspondientes, doy por aprobada la redacción final del documento escrito para que prosiga con los trámites correspondientes a la sustentación de la Defensa oral.

Pamela Catherine Flores Naranjo, PhD

DIRECTOR

DECLARACIÓN DE AUTORÍA

Yo, Carlos Andrés Cueva Anchapaxi, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

La Escuela Politécnica Nacional puede hacer uso de los derechos correspondientes a este trabajo, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

Carlos Andrés Cueva Anchapaxi

ÍNDICE DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

LISTA DE ANEXOS

RESUMEN

La Infraestructura como Código (IaC) cumple un rol fundamental en la práctica DevOps y es vital asegurar que los scripts de IaC están libres de defectos de seguridad, de tal manera, evitar introducir debilidades de seguridad en la infraestructura tecnológica o sistemas gestionados. Para cumplir este objetivo es necesario identificar los factores asociados con estas anomalías en un script de IaC y corregir cada uno. Este trabajo ejecuta análisis correlacional entre métricas de procesos, relacionadas con defectos en archivos de código fuente, y defectos de seguridad en scripts de IaC con el propósito de determinar el nivel de correlación entre estas variables y conocer si las métricas de proceso son factores correlacionados con defectos de seguridad en scripts Puppet. Repositorios Puppet de GitHub son recolectados para extraer las métricas asociadas a los scripts y los defectos de seguridad son identificados usando análisis estático de código. Los resultados permiten determinar que las métricas de procesos número de revisiones, número de autores de revisiones, número de líneas modificadas, número promedio de días entre revisiones, número promedio de líneas modificadas por revisión y edad del script, y defectos de seguridad, como administrador por defecto, contraseña vacía, secretos codificados, comentarios sospechosos, acceso sin restricción, HTTP sin TLS y algoritmo de criptografía débil, presentan correlación muy débil o despreciable, con coeficiente de correlación Spearman menor o igual a 0.23. Este hallazgo puede indicar que el proceso de desarrollo de scripts de IaC es diferente al proceso de desarrollo de código fuente.

Palabras clave: Métricas de Proceso. Defectos de Seguridad. Infraestructura como Código (IaC).

ABSTRACT

Infrastructure as Code (IaC) is fundamental in DevOps practice and is important to make sure that IaC scripts are security defects free so we can avoid introduce security weakness to the technological infrastructure or systems managed. To accomplish this goal is necessary to identify associated factors with these kinds of anomalies and fix each one. This work executes a correlational analysis with process metrics, related to software defects in source code, and security defects in IaC scripts, so we can find out the correlation level between these variables and know whether process metrics are factors correlated to security defects in scripts Puppet. Puppet repositories are downloaded from GitHub to collect process metrics and security defects are identified using static code analysis. Results allow to know that process metrics such number of revisions, number of authors revisions, number of modified lines, average number of modified lines per revision, scripts age, and security defects such admin by default, empty password, hard-coded secret, invalid IP address binding, suspicious comment, HTTP without TLS and weak cryptography algorithm, are very low correlated, with Spearman coefficient equal or less than 0.23. This finding may suggest that scripts IaC development process is different than source code development process.

Keywords: Process Metrics. Security Defects. Infrastructure as Code (IaC).

1. INTRODUCCIÓN

Infraestructura como Código (IaC por sus siglas en inglés) es la gestión de la infraestructura tecnológica (aprovisionamiento de máquinas virtuales, servidores, redes, instalación y configuración de paquetes de software) a través de código (scripts) y uso de prácticas tradicionales de la Ingeniería de Software, como la gestión de calidad de código, pruebas, control de versión (Guckenheimer, 2017), (Sharma, Fragkoulis, & Spinellis, 2016). IaC es fundamental en la práctica DevOps, especialmente, para la entrega continua de software. Varias organizaciones grandes, como Netflix, GitHub, Mozilla, usan tecnologías de IaC populares, como Puppet[1] o Chef[2], para automatizar el aprovisionamiento de máquinas virtuales, administrar bases de datos, gestionar cuentas de usuario, instalar paquetes de software (Rahman, Mahdavi-Hezaveh, & Williams, 2018). Por otro lado, las métricas de proceso son atributos del proceso de desarrollo de software que pueden ser medidos y expresados numéricamente. Por ejemplo, el tiempo que toma completar una tarea o el número de ocurrencias de un evento en particular (Sommerville, 2011), son utilizadas para describir el cambio del software a través del tiempo (Madeyski & Jureczko, 2015) y mejorar el proceso de desarrollo y mantenimiento del software (Kan, 1995).

Similar al código fuente, que tiene defectos (errores, vulnerabilidades, deuda técnica), los scripts de IaC también pueden presentar estas anomalías, ya que están propensos al error humano y prácticas erróneas de codificación durante el proceso de desarrollo (Sharma, Fragkoulis, & Spinellis, 2016). En el trabajo titulado Security Smells in Infrastructure as Code Scripts (Rahman, Rahman, Parnin, & Williams, 2019), los autores identificaron varios defectos de seguridad que pueden ser encontrados en scripts de IaC, y determinaron que no todos ellos presentan defectos. Este resultado despierta el interés por conocer cuales características, representadas en forma de métricas, están relacionadas con los defectos de seguridad. Conocer las características en común puede ayudar a gestionar con mayor eficiencia las actividades de aseguramiento de calidad de código (pruebas, inspección, análisis estático, etc.).

Estudios previos, (Misirli, Murphy, Zimmermann, & Basar, 2011), (Madeyski & Jureczko, 2015), han identificado métricas de proceso, como número de revisiones, número de cambios, número de desarrollares, edad del archivo, tiempo promedio entre ediciones,

[1] https://puppet.com

[2] https://chef.io

relacionadas con defectos en artefactos de software, escritos con un lenguaje de propósito general, y han construido modelos de predicción de defectos de software en base a estas métricas. Este mismo enfoque resulta útil en los scripts de IaC. Sin embargo, las métricas de proceso evaluadas pueden no ser aplicables en estos artefactos de software, ya que son escritos con un lenguaje específico de dominio (Kanies, 2012), el cual tiene sintaxis y semántica diferente.

Este trabajo propone analizar la correlación entre métricas de proceso, como, número de revisiones, número de autores de revisiones, número de líneas modificadas, tiempo promedio entre revisiones, número promedio de líneas modificadas por revisión y edad del script, y defectos de seguridad en scripts de Infraestructura como Código (IaC) para determinar si las mismas pueden ser usadas como indicadores de defectos de seguridad. Scripts de IaC y métricas de proceso relacionadas a estos, pueden ser obtenidos de repositorios de código, con software de control de versión, disponibles en plataformas como GitHub[3]. Los repositorios de código contienen información valiosa sobre el código, personas y procesos involucrados en el desarrollo de un producto de software, y han sido objeto de estudio en varios trabajos de investigación (Munaiah, Kroh, Craig, & Nagappan, 2017).

1.1. Pregunta de investigación

Este trabajo plantea la pregunta de investigación: ¿Cuál es la correlación entre métricas de proceso y defectos de seguridad en scripts de Infraestructura como Código?

1.2. Objetivo general

Analizar la correlación entre métricas de proceso y defectos de seguridad en scripts de Infraestructura como Código.

1.3. Objetivos específicos

- Construir un conjunto de datos de repositorios de scripts de Infraestructura como Código para obtener métricas de proceso y defectos de seguridad.
- Identificar defectos de seguridad en scripts de Infraestructura como Código mediante análisis estático de código.
- Determinar el grado de relación que existe entre métricas de proceso y defectos de seguridad en scripts de Infraestructura como Código.

[3] https://github.com

2. MARCO TEÓRICO

Esta sección presenta fundamentos sobre scripts de Infraestructura como Código, métricas de proceso y trabajos académicos relacionados.

1.4. Scripts de Infraestructura como Código

Los scripts de la tecnología de Infraestructura como Código que serán estudiados en este trabajo son los scripts Puppet. Puppet es una de las tecnologías de IaC más populares. Hay repositorios de código con gran cantidad de scripts Puppet disponibles en GitHub y sus scripts han sido objeto de estudio en varios trabajos de investigación. De acuerdo con (Puppet, 2019), un script Puppet es denominado manifiesto, con la extensión de archivo pp, y es usado para definir clases y recursos. La clase es utilizada para organizar recursos, y permitir la reutilización de código.

El recurso es la unidad fundamental en el modelamiento de la configuración de un sistema y es utilizado para describir el estado deseado de un elemento (servicio, paquete, archivo, nodo, operación) del sistema. Por ejemplo, la Figura 1 presenta el código Puppet utilizado para definir recursos que tienen como objetivo actualizar los paquetes de software del sistema operativo, instalar el paquete apache2 e iniciar la ejecución de este último. Como puede ser identificado en la Figura 1, un recurso es definido por el tipo, título o nombre, atributos y valores para especificar su estado.

```
1   # recurso para ejecutar comando
2   exec { 'apt-update':                    # recurso exec
3     command => '/usr/bin/apt-get update'  # atributo command
4   }
5
6   # recurso para definir instalar un paquete de software
7   package { 'apache2':                    # recurso package
8     require => Exec['apt-update'],        # atributo require
9     ensure => installed,                  # atributo ensure
10  }
11
12  # recurso para definir el estado de un servicio
13  service { 'apache2':                    # recurso service
14    ensure => running,                    # atributo ensure
```

Figura 1 - Código Puppet para instalar paquete de software (Mitchell, 2014)

Otro fragmento de código Puppet puede ser observado en la Figura 2, el cual es utilizado para aceptar conexiones de cualquier dirección IP en un servidor PostsgreSQL[4] y asignar la contraseña del usuario postgres. De acuerdo con (Rahman, Parnin, & Williams, 2019), las configuraciones aceptar conexiones desde cualquier dirección IP y codificar una contraseña en el archivo, son consideradas debilidades de seguridad en un Script Puppet. Estas debilidades de seguridad son algunas de las debilidades que son consideradas defectos de seguridad en el presente estudio y la lista completa de defectos es presentada en el capítulo 3.

```
1  class { 'postgresql::server':
2    ip_mask_allow_all_users    => '0.0.0.0/0',
3    postgres_password          => 'password1234',
4  }
```

Figura 2 - Código Puppet para configurar PostgreSQL
(Puppetforge, 2019)

1.5. Métricas de proceso

Una métrica de proceso es un atributo o característica del proceso de desarrollo de software que puede ser medida y expresada numéricamente (Sommerville, 2011). Las métricas de proceso son utilizadas para describir el cambio del software a través del tiempo (Madeyski & Jureczko, 2015) y mejorar el proceso de desarrollo y mantenimiento del software (Kan, 1995). Conocer los aspectos del desarrollo relacionados o asociados con los defectos de software es útil para investigar más detalle sobre ellos y corregir cada uno a tiempo, ya que el costo por arreglar defectos incrementa a medida que el proceso avanza (Stecklein, et al., 2004).

Estudios previos han defino métricas de proceso y evaluado su correlación con defectos, como, vulnerabilidades, deuda técnica, errores, en software escrito con lenguaje de propósito general (Java, C, etc.). Por ejemplo, (Meneely & Williams, 2009), estudiaron métricas relacionadas con la actividad de desarrollares de software y su relación con el número de vulnerabilidades en el kernel de Red Hat Enterprise Linux 4; y determinaron que la probabilidad de introducir debilidades de software en un archivo editado por más de 9 desarrolladores es 16 veces mayor que la probabilidad de introducir vulnerabilidades en un archivo editado por menor número de desarrolladores.

[4] https://www.postgresql.org/

(Misirli, Murphy, Zimmermann, & Basar, 2011) analizaron la correlación entre métricas de proceso y defectos en dos versiones beta del software Eclipse y determinaron que la edad del archivo, número de autores de commits, tiempo promedio entre ediciones, número de líneas promedio modificadas, última fecha de edición y número de ediciones, pueden ser usadas como indicadores independientes de defectos de software.

(Illes-Seifert & Barbara, 2010), estudiaron la relación entre características históricas de un archivo y el número de defectos en nueve productos de Java open source, e identificaron que el número de defectos incrementa junto al número de cambios en el archivo, aplicados casi al terminar el periodo antes de una entrega (release). También identificaron que el número de autores de cambios y la frecuencia de cambios están correlacionados con el número de defectos en los archivos de los proyectos analizados.

(Madeyski & Jureczko, 2015) realizaron un estudio extenso sobre métricas de proceso definidas y utilizadas en varios trabajos de investigación, e identificaron que las métricas más utilizadas para construir modelos de predicción son el número de revisiones (o cambios), número de defectos en versiones previas, número de autores de revisiones y número de líneas modificadas, porque presentan correlación alta con el número de defectos y mejoran el rendimiento de los modelos de predicción.

En base a los resultados de los trabajos revisados, las siguientes métricas de proceso serán recolectadas de los repositorios de scripts Puppet para analizar su correlación con el número de defectos de seguridad: número de revisiones, número de autores de revisiones, número de líneas modificadas, número promedio de días entre revisiones, número promedio de líneas modificadas por revisión y edad del script.

1.6. Trabajos relacionados

El presente trabajo académico está relacionado con trabajos que estudiaron defectos y fallas de seguridad en scripts de IaC Puppet. Por ejemplo, (Rahman, Mahdavi-Hezaveh, & Williams, 2018) ejecutaron un mapeo sistemático sobre trabajos de investigación en IaC, con el propósito de identificar las áreas de investigación en este dominio y proponer líneas de investigación. Revisaron alrededor de 31500 publicaciones, de ACM Digital Library, IEEE Xplore, Science Direct, Springer Link, Wiley Online, e identificaron 32 trabajos sobre Infraestructura como Código revisados por pares. Los autores identificaron cuatro temas de investigación: marcos de trabajo/herramientas para IaC, adopción de IaC, estudios empíricos relacionados a IaC, pruebas en IaC. El resultado de este trabajo permitió

determinar la necesidad de investigación sobre defectos y fallas de seguridad en scripts de IaC, ya que estas anomalías pueden causar problemas con impacto alto en ambientes en donde IaC es utilizada, como el despliegue y desarrollo de software.

(Sharma, Fragkoulis, & Spinellis, 2016) identificaron 13 smells de implementación (relacionados con el estilo, formato, convención de nombres y sangría del código) y 11 smells de diseño (relacionados con el diseño del módulo o estructura de un proyecto) en scripts Puppet, que no están de acuerdo con las prácticas recomendadas para código de configuración. Primero, recopilaron 4,621 repositorios Puppet de GitHub, usando GHTorrent para obtener información de los repositorios. Luego, aplicaron herramientas de análisis estático de código sobre los scripts para identificar los smells de implementación y diseño. Los smells de implementación fueron identificados con la herramienta Puppet-Lint. Los smells de diseño fueron identificados por la herramienta Puppeter, desarrollada por los autores, la cual detecta problemas relacionadas con el diseño del módulo o configuración del proyecto; estos problemas fueron definidos de acuerdo con la guía de estilo de Puppet, entradas de blogs, y videos de charlas técnicas. El resultado de este trabajo permitió identificar que los smell de implementación más comunes son alineamiento no apropiado, uso no apropiado de comillas, declaración de gran tamaño. En el caso de smells de diseño, aquellos con mayor frecuencia son falta de modularizarían y abstracción multifacética. También determinaron que estos dos tipos de smells están correlacionados entre si (coeficiente de correlación Spearman igual a 0.66).

(Rahman & Williams, 2018) caracterizaron scripts Puppet defectuosos mediante un grupo de palabras de texto, utilizando las técnicas de minería de texto, bag-of-words (BOW) y term frequency-inverse document frequency (TF-IDF), y construyeron un modelo de predicción, basado en las palabras obtenidas por cada técnica, usando el algoritmo Random Forest. Primero, recopilaron repositorios de software open source de OpenStack, Wikimedia, Mozilla, que contienen scripts Puppet, y generaron 4 conjuntos de datos, uno para cada organización que mantiene los repositorios. Luego, clasificaron los commits de los repositorios que señalan la gestión de defectos en scripts Puppet, con el propósito de identificar los scripts relacionados, y clasificarlos como defectuosos. Después, aplicaron las técnicas de minería de texto para caracterizar los scripts defectuosos en base a las palabras más representativas. A continuación, aplicaron Principal Component Analysis sobre estas palabras para identificar aquellas palabras que presentan mayor correlación con un script defectuoso y pueden ser utilizadas en la construcción de un modelo de predicción. Para establecer el contexto de estas palabras, y de esta manera, entender o

definir el tipo de defecto, aplicaron la técnica de análisis cualitativo Strauss-Corbin Grounded Theory (SGT), la cual permite caracterizar palabras. Derivaron 3 conceptos de alto nivel para los grupos de palabras que representan al script con defectos: operaciones en sistemas de archivo, aprovisionamiento de infraestructura, manejo de cuentas de usuario. Luego, construyeron 2 modelo de predicción y evaluaron cada uno con la técnica de validación cruzada 10x10. El rendimiento de los modelos fue medido con las métricas medida-F y área bajo la característica operativa del receptor, obteniendo como resultado, que el rendimiento del modelo de predicción en base a la técnica BOW es similar al rendimiento del modelo de predicción en base a la técnica de TF-IDF. Las medidas de área bajo la característica operativa del receptor son 0.74, 0.71, and 0.73 para Mozilla, Openstack y Wikimedia, y las medidas de medida-F son 0.72, 0.74, and 0.70 para Mozilla, Openstack y Wikimedia, respectivamente.

(Rahman & Williams, 2019) identificaron 10 propiedades de código correlacionadas con scripts Puppet defectuosos, y construyeron modelos de predicción de scripts defectuosos, basados en estas características, usando 5 algoritmos clasificadores (Classification and Regression Trees, K Nearest Neighbor classification, Logistic Regression, Naive Bayes classification, Random Forest). Primero, recopilaron repositorios de software open source de OpenStack, Wikimedia, Mirantis, Mozilla, y generaron 4 conjuntos de datos, uno por cada organización que mantiene los repositorios. Luego, clasificaron los commits que señalan gestión de defectos en scripts Puppet para identificar los scripts defectuosos. Clasificaron los defectos en 9 categorías usando la técnica Orthogonal Defect Classification (ODC). Luego, identificaron las características de código mediante el análisis cualitativo Constructivist Grounded Theory sobre los commits relacionados a defectos. A continuación, identificaron las propiedades de código más representativas, mediante Principal Component Analysis (PCA). Después, construyeron los modelos de predicción con la herramienta API Scikit Learn, con las propiedades identificadas con PCA; para el conjunto de datos de Mirantis y Mozilla usaron una propiedad de código, y para Openstack y Wikimedia usaron dos propiedades de código. Para entrenar, evaluaron los modelos usando validación cruzada 10x10, y midieron el rendimiento de cada modelo con 4 métricas: precisión, sensibilidad (recall), medida-F y área bajo la característica operativa del receptor. También, construyeron modelos de predicción, adicionales, usando otros factores relacionadas a defectos en scripts Puppet y defectos en código fuente, los cuales fueron identificados en estudios previos, como, smells de implementación en scripts Puppet, métricas de proceso en código fuente, características basadas en texto (bag-of-words) en scripts Puppet, con el propósito de comparar el rendimiento de los modelos de

predicción basados en las propiedades de código identificadas en su trabajo versus modelos de predicción basados en otros factores.

El rendimiento de cada uno de los modelos de predicción basados en características de código es diferente (respecto a cada métrica definida) en cada uno de los conjuntos de datos (Mozilla, Mirantis, Wikimedia, Openstack), con la excepción del modelo de predicción de Bayes Naive, el cual tiene el mejor rendimiento medido por precisión (0.80 a 0.85) en todos los conjuntos de datos. Por otro lado, los modelos basados en características de código tienen mejor rendimiento que los modelos de predicción que usan características de texto en todos los conjuntos de datos. Este resultado es similar en la comparación del rendimiento de los modelos de predicción basados en características de código respecto al rendimiento de los modelos de predicción basados en smells de implementación, con la excepción de que estos últimos tienen mejor sensibilidad (Logistic Regresion con 0.97, Random Forest con 0.96 y Classification and Regression Trees con 0.93), por lo cual son recomendados cuando el número alto de falsos positivos no son un inconveniente. Por otro lado, los modelos de predicción basados en métricas de proceso presentan un mejor rendimiento, de acuerdo con ciertas métricas, en algunos conjuntos de datos. Sin embargo, estos resultados no son concluyentes ya que el mismo modelo tiene diferente rendimiento en cada conjunto de datos. Es por esta razón, que los autores recomiendan construir modelos de predicción combinando características de código y métricas de procesos.

(Rahman, Parnin, & Williams, 2019) identificaron 7 smells de seguridad en scripts Puppet, mediante análisis cualitativo, construyeron una herramienta de análisis estático para identificar cada uno, y presentaron estrategias de mitigación. Un smell de seguridad es definido como un patrón de código que señala de debilidad de seguridad y que puede permitir una brecha de seguridad. Primero, recopilaron 1,726 scripts Puppet de 74 repositorios de Mozilla, Openstack y Wikimedia. Luego, aplicaron la técnica de análisis cualitativo Descriptive Coding para identificar los smells de seguridad: admin by default, empty password, hard-coded secret, invalid ip address binding, suspicious comment, use of http without tls, use of weak cryptography algorithms. A continuación, construyeron la herramienta de análisis estático Security Linter for Infrastructure as Code scripts, o SLIC, en base a la herramienta Puppet-Lint. Después, evaluaron el rendimiento de esta herramienta, analizando 250 scripts Puppet seleccionados al azar del conjunto de scripts recolectados previamente, y los resultados de la herramienta fueron comparados con la inspección manual realizada por los autores, obteniendo que la sensibilidad de SLIC ≥ 0.95,

lo cual permitió concluir que SLIC permite identificar la mayoría de smells de seguridad en un script Puppet.

(Rahman, Rahman, Parnin, & Williams, 2019) aplicaron la misma metodología de (Rahman, Parnin, & Williams, 2019) para identificar smells de seguridad en scripts Chef y Ansible: admin by default, empty password, hard-coded secret, invalid ip address binding, suspicious comment, use of http without tls, use of weak cryptography algorithms. De los 9 smells de seguridad definidos, 6 smells pueden ser encontrados en scripts Ansible, 8 smells pueden ser encontrados en scripts Chef y 7 smells en scripts Puppet.

Los trabajos de investigación de (Sharma, Fragkoulis, & Spinellis, 2016), (Rahman & Williams, 2018), (Rahman & Williams, 2019), estudiaron defectos que afectan la mantenibilidad de código en scripts de IaC. (Rahman & Williams, 2018), (Rahman & Williams, 2019), construyeron modelos de predicción en base a factores intrínsecamente correlacionados con un script Puppet con defectos (los factores utilizados fueron extraídos de scripts defectuosos). (Rahman & Williams, 2019) construyeron modelos de predicción en base a métricas de proceso, identificas como factores correlacionadas con defectos en archivos de código fuente. Sin embargo, estas métricas de proceso no necesariamente están correlacionadas con defectos en scripts IaC, ya que este tipo de artefacto de software no es creado con el mismo tipo de lenguaje de programación, y el proceso de desarrollo para crear un archivo de código fuente y un script de IaC puede no ser igual. Los defectos de seguridad que pueden ser encontrados en scripts de IaC han sido identificados en los estudios de (Rahman, Parnin, & Williams, 2019), (Rahman, Rahman, Parnin, & Williams, 2019). En este último trabajo identificaron que no todos los scritps de IaC presentan estas anomalías, por lo cual es de interés identificar los factores en común.

El presente trabajo académico aporta al estado del arte con el análisis correlacional entre métricas de procesos y defectos de seguridad en scripts de IaC. El resultado permitirá identificar cuales métricas de proceso presentan mayor correlación, determinar si son factores comunes en los scripts de IaC con defectos de seguridad, conocer si las métricas de procesos identificadas como factores correlacionados con defectos en archivos de código también presentan correlación con scripts de IaC que tienen defectos de seguridad. Las métricas de proceso que presentan mayor correlación pueden ser utilizadas en un trabajo futuro para construir modelos de predicción de scripts de IaC con defectos de seguridad, y de esta manera abordar otros tipos de defectos, diferentes a los defectos que

afectan a la mantenibilidad de código, estudiados por (Sharma, Fragkoulis, & Spinellis, 2016), (Rahman & Williams, 2018), (Rahman & Williams, 2019).

3. METODOLOGÍA

El objetivo general de este trabajo es analizar la correlación entre métricas de proceso y defectos de seguridad en scripts de Infraestructura como Código. Es por esta razón, que la investigación correlacional es tomada como metodología de referencia. La investigación correlacional es utilizada para analizar la asociación lineal o correlación entre dos variables cuantitativas, sin intención de manipular, controlar estas variables; y en el caso que exista asociación lineal, los resultados de una variable pueden ser usados para predecir los resultados de la otra variable (Frederick & Larry, 2013). En este estudio las métricas de proceso y defectos de seguridad son las variables cuantitativas, y son obtenidas de repositorios de código disponibles en GitHub, ajenos al autor, que no serán modificados.

3.1. Investigación correlacional

Fundamentalmente, la investigación correlacional es ejecutada en dos fases: recolección de datos y análisis correlacional (Frederick & Larry, 2013), (Price, Jhangiani, & I-Chant, 2019). La recolección de datos implica la recolección o medición de variables cuantitativas de los individuos que pertenecen a un grupo, sin manipular estas y sin modificar el ambiente de donde son obtenidas. El análisis correlacional corresponde al cálculo e interpretación del coeficiente de correlación, el cual es usado para medir la asociación lineal entre las variables.

3.2. Recolección de datos

Las variables analizadas en la correlación son observadas o medidas en su entorno, sin la intención de controlar o manipular cada una de estas. En este estudio, observar o medir, corresponde a recopilar las métricas de proceso de los repositorios GitHub e identificar y cuantificar el número de defectos de seguridad en scripts Puppet. El resultado de esta actividad es un conjunto de medidas de las dos variables (usualmente denominadas X, Y), el cual es la entrada para calcular el coeficiente de correlación en la fase de análisis correlacional. Las medidas, por lo general, son presentadas en tablas o gráfico de dispersión (X: abscisa, Y: ordenada). Por ejemplo, la Figura 3 muestra un conjunto de medidas de dos variables de un estudio correlacional. Las variables son Family Income y Student´s Average Grade, y la muestra es de 14 estudiantes de High School.

En la Figura 3 es posible identificar la relación entre las variables, cuando una variable crece, la otra variable también crece. A pesar de que la representación gráfica es útil, para identificar la relación entre las variables, cuando el tamaño de la muestra es grande resulta

difícil distinguir esta, y es más apropiado usar el análisis correlacional para evaluar la relación entre las variables.

Person	Family Income (in $1000)	Student's Average Grade
A	31	72
B	38	86
C	42	81
D	44	78
E	49	85
F	56	80
G	58	91
H	65	89
I	70	94
J	90	83
K	92	90
L	106	97
M	135	89
N	174	95

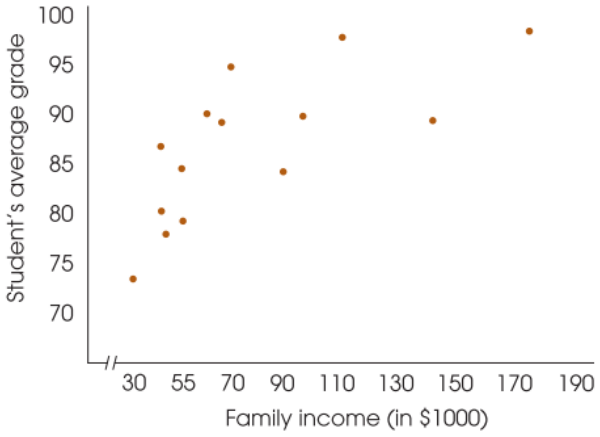

Figura 3 - Ejemplo de medidas ingreso económico y calificaciones promedio de estudiantes de un estudio correlacional.
(Frederick & Larry, 2013)

3.3. Análisis correlacional

De acuerdo con (Rummel, 1976) y (Frederick & Larry, 2013), el objetivo principal de la ciencia es establecer relaciones entre variables o características, y la correlación es una de las técnicas de la estadística que permite identificar si hay relación entre variables de los individuos de un grupo o población. El análisis correlacional es utilizado para medir y describir la relación entre dos variables, las cuales son medidas u observadas en cada uno de los individuos que pertenecen al grupo.

Los resultados de un estudio correlacional permiten mostrar si existe relación o asociación entre dos variables (varían en conjunto), pero no explicar la relación, es decir, no es útil para identificar una relación causa-efecto. Una aplicación común de la correlación es la predicción (Frederick & Larry, 2013). Si existe asociación línea entre dos variables, el valor de una variable puede ser utilizado para predecir el resultado de la otra variable. Por ejemplo, estudios previos construyeron modelos de predicción de artefactos de software defectuosos, usando factores correlacionados con el número de defectos de software (Madeyski & Jureczko, 2015).

3.3.1. Coeficiente de correlación

El coeficiente de correlación es un número entre -1 y 1, utilizado para describir y medir dos características de la relación entre las variables (Frederick & Larry, 2013).

1. **Dirección de la relación**: el signo del coeficiente de correlación indica la dirección de la relación de las variables. El signo positivo señala correlación positiva, lo cual significa que las dos variables crecen o decrecen en conjunto. El signo negativo indica que la correlación negativa, lo cual significa que las variables crecen o decrecen en direcciones opuestas (variable X crece, variable Y decrece, y viceversa).
2. **Consistencia de la relación:** la magnitud del coeficiente mide la consistencia de la relación. Cuando las variables X, Y crecen o decrecen en conjunto o en direcciones opuestas a una tasa de cambio constante, la relación es consistente y las variables tienen relación lineal. Sin embargo, esta tasa de cambio puede no ser constante, y la magnitud del coeficiente de correlación permite interpretar el nivel de consistencia de la relación. La interpretación de la magnitud es presentada en la sección Interpretación del coeficiente de correlación.

El método Pearson o correlación Pearson es utilizado para calcular el coeficiente de correlación. Hay otros métodos para calcular la correlación (por ejemplo, Spearman), que son aplicaciones particulares del método Pearson (Frederick & Larry, 2013). Este método es utilizado para medir el nivel y dirección de la relación lineal entre dos variables, asumiendo que las distribuciones de estas variables son similares a una distribución normal. La correlación es calculada como la relación entre el nivel de variación en conjunto de las dos variables y el nivel de variación de cada variable por separado (Frederick & Larry, 2013). El nivel de variación en conjunto de las dos variables es calculado mediante la covarianza, y el nivel de variación de cada variable es calculado como la desviación estándar de cada medida respecto al promedio. De acuerdo con estas definiciones la fórmula para calcular la correlación Pearson es la siguiente.

$$r = \frac{Cov(X,Y)}{\sigma_X \sigma_Y}$$

Por otro lado, el método Spearman o correlación Spearman es utilizado en los casos que las variables son ordinales (IRI, 2019) (Frederick & Larry, 2013) o las distribuciones de las muestras de las variables son diferentes a una distribución normal. También puede ser utilizado con variables continuas, y es una alternativa al método Pearson cuando la consistencia de la relación entre las variables no es constante y el propósito de la

investigación es conocer si existe alguna forma de relación entre estas; por ejemplo, la relación es continua, o en una dirección (similar a una función monótona), y no es estrictamente lineal. La diferencia más destacada respecto al método Pearson es el grado de sensibilidad a los valores atípicos o outliers, siendo el método Spearman más resistente a los valores atípicos que el método Pearson (IRI, 2019). Para calcular la correlación Spearman con medidas de variables continuas, es necesario ordenar las medidas de X, Y, dentro de la muestra respectiva y aplicar la fórmula de Pearson con las nuevas variables ordinales (Frederick & Larry, 2013).

Con los métodos Pearson y Spearman es factible identificar si dos variables están relacionadas. Aunque la mayoría de las distribuciones de variables en la naturaleza son semejantes a la distribución normal (Frederick & Larry, 2013), y es por esa razón, que varios trabajos de investigación asumen esta sentencia como un hecho, aplicando Pearson para calcular la correlación, es recomendado evaluar la distribución de la muestra para determinar cuál método de correlación usar. Algunas de las técnicas utilizadas para determinar o comparar distribuciones son: histograma (visualizar si la distribución es similar a la distribución normal), gráfico de cuantiles o Q-Q (NIST/SEMATECH, 2012), prueba de normalidad (Wilk & Shapiro, 1965).

3.3.2. Interpretación del coeficiente de correlación

De acuerdo con (Rummel, 1976), (Frederick & Larry, 2013), el cuadrado del coeficiente de correlación, denominado coeficiente de determinación, mide la porción de variabilidad que tienen las dos variables en conjunto, y es utilizado para interpretar la consistencia de la correlación. Considerando como ejemplo un caso en el cual el coeficiente de correlación entre dos variables es 0.5, y en consecuencia el coeficiente de determinación es 0.25. La interpretación de este resultado corresponde a que la variación del 25% de los valores de una variable está asociada a la variación de la segunda variable.

Tabla 1 - Intervalos para interpretar el coeficiente de correlación

COEFICIENTE DE CORRELACIÓN (r)	**COEFICIENTE DE DETERMINACIÓN (r^2)**	**ASOCIACIÓN LINEAL o CORRELACIÓN**
$\|r\| < 0.3$	$r^2 < 0.09$	Muy débil
$0.3 \leq \|r\| < 0.5$	$0.09 \leq r^2 < 0.25$	Débil
$0.5 \leq \|r\| < 0.7$	$0.25 \leq r^2 < 0.49$	Moderada
$\|r\| \geq 0.7$	$r^2 \geq 0.49$	Fuerte

Fuente: (Haldun, 2018), (Mukaka, 2012), (Mindrila & Balentyne, 2019), (Moore, Notz, & Flinger, 2013)

Varios investigadores han propuesto interpretar la consistencia de la correlación definiendo la magnitud del coeficiente de correlación en intervalos, como puede ser observado en la Tabla 1. La definición de los intervalos depende del área de investigación, sin embargo, los intervalos presentados en la Tabla 1 pueden ser utilizados como referencia en diferentes áreas.

3.3.3. Prueba de hipótesis

Por lo general, la correlación es calculada con medidas de una muestra, con el propósito de responder preguntas sobre la correlación de la población. Sin embargo, cuando la muestra es usada, la correlación calculada puede ser representativa o un error de muestreo (Frederick & Larry, 2013). Por lo tanto, es recomendado usar la técnica estadística prueba de hipótesis para identificar uno de estos dos resultados. Fundamentalmente, la prueba de hipótesis evalúa dos suposiciones o hipótesis sobre la población para determinar cuál de estas es respaldada por los datos de la muestra (Ogee, Ellis, Scibilia, Pammer, & Steele, 2015).

- Hipótesis nula (Ho): no existe correlación en las dos variables de la población.
- Hipótesis alternativa (H1): existe correlación en las dos variables de la población.

La primera hipótesis formulada sobre la población es denominada hipótesis nula, y define que no existe correlación en la población. La segunda hipótesis es la hipótesis alternativa, la cual intenta contradecir a la primera, por lo tanto, define que existe correlación. Cuando la hipótesis alternativa define explícitamente que la correlación es negativa o positiva, la prueba de hipótesis es denominada prueba de hipótesis direccional o de una cola (Frederick & Larry, 2013). Para aceptar una de las hipótesis, dos variables son comparadas: el nivel de significación o nivel alfa (α) y la probabilidad de la estadística correlación (p-valor o p-value). De acuerdo con (Frederick & Larry, 2013), la definición de estos elementos es la siguiente:

- Nivel alfa: determina la probabilidad de rechazar la hipótesis nula cuando en realidad es verdadera. El nivel alfa es utilizado para establecer el límite que separa a las estadísticas con probabilidad alta de las estadísticas con probabilidad baja. La elección del nivel alfa es arbitraria y depende del investigador señalar el nivel alfa antes de iniciar la investigación. Por lo general los niveles de alfa son α = .05 (5%), α = .01 (1%). Por ejemplo, el nivel alfa α = .05, indica que el 95% de las estadísticas son comunes (valores centrales), y 5 % de las estadísticas son muy improbables (valores extremos). Los valores extremos forman la región denominada región crítica.

- p-valor: probabilidad de obtener una estadística correlación en la región crítica cuando la hipótesis nula es verdadera. Los valores de probabilidad están registrados en tablas estadísticas (Frederick & Larry, 2013), o pueden ser calculadas con paquetes de software estadístico.

Para aceptar la hipótesis nula el p-valor debe ser mayor que el nivel alfa (p-valor > α). Por otra parte, para rechazar la hipótesis nula, y en consecuencia aceptar la hipótesis alternativa, el p-valor debe ser menor que el nivel alfa (p-valor < α). En los trabajos de investigación correlacional, la correlación entre variables de una muestra es representativa de la población cuando la hipótesis nula es rechazada (Frederick & Larry, 2013).

4. ANÁLISIS CORRELACIÓN ENTRE MÉTRICAS DE PROCESO Y DEFECTOS DE SEGURIDAD

Este capítulo detalla la ejecución del análisis correlacional entre métricas de proceso y defectos de seguridad en scripts Puppet. La Figura 4 muestra de forma resumida el proceso ejecutado, el cual está formado por 4 fases. La primera fase corresponde a la recolección de repositorios que contienen scripts Puppet de la plataforma GitHub. A continuación, la extracción de métricas de procesos es realizada. Después, la identificación de defectos de seguridad en scripts Puppet es ejecutada usando análisis estático de código. El resultado de estas dos últimas etapas genera el conjunto de datos de medidas de métricas de proceso y número de defectos de seguridad, las cuales son usadas en la última etapa que corresponde a el análisis correlacional entre métricas de procesos y defectos de seguridad.

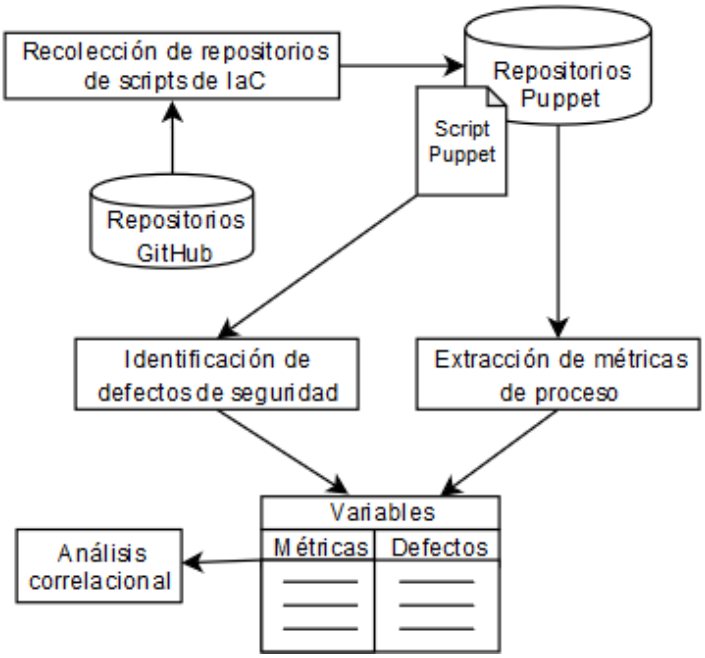

Figura 4 - Análisis correlacional entre métricas de proceso y defectos de seguridad

4.1. Recolección de repositorios de scripts de IaC

La recolección de repositorios de scripts de IaC corresponde a descargar repositorios Puppet de GitHub, los cuales son la fuente para obtener los scripts Puppet y las métricas de procesos asociadas a cada uno. GitHub hospeda millones de repositorios, y muchos de ellos no pertenecen a proyectos de Ingeniería de Software, en su lugar, son producto de proyectos personales, proyectos académicos o pruebas. Por lo tanto, varios criterios son definidos para filtrar aquellos que no aportan con información sobre proyectos de Ingeniería

de Software, y de tal manera, construir un conjunto de datos de repositorios de scripts de IaC con información relevante.

- **Criterio 1:** el repositorio debe ser de lenguaje Puppet. En GitHub el lenguaje de programación de un repositorio es asignado en base al lenguaje con el cual están escritos la mayoría de los archivos presentes. En este estudio, los scripts Puppet son el objeto de estudio, por lo cual solo repositorios de lenguaje Puppet son considerados.
- **Criterio 2:** el repositorio deber ser público y descargable. La detección de defectos de seguridad es realizada mediante análisis estático de código, por lo cual es necesario tener una copia de los scripts Puppet y las métricas de proceso son obtenidas del registro de cambios o revisiones, y en este caso corresponde al historial de commits del software de control de versión Git[5], utilizado por GitHub.
- **Criterio 3:** el repositorio debe tener al menos 2 commits por mes en promedio durante su periodo de vida. El periodo de vida está definido entre la fecha del primer commit y la fecha del último commit, disponible en el instante en el cual este estudio fue ejecutado, en la rama principal del repositorio. (Munaiah, Kroh, Craig, & Nagappan, 2017) determinaron que 2 commits por mes en promedio es el número mínimo de commits que caracteriza a un proyecto de Ingeniería de Software que evoluciona y es mantenido.
- **Criterio 4:** el repositorio debe ser mantenido por más de 9 contribuidores. Una característica de un proyecto de Ingeniería de software es la colaboración, expresada por el número de contribuidores (Munaiah, Kroh, Craig, & Nagappan, 2017). El estudio de (Meneely & Williams, 2009) fue tomado como referencia para establecer este criterio, en el cual determinaron que la probabilidad de introducir debilidades de seguridad incrementa cuando un archivo es editado por más de 9 desarrolladores.
- **Criterio 5:** al menos 11% de los archivos del repositorio deben ser scripts Puppet. (Jiang & Adams, 2015) observaron que el porcentaje de scripts de IaC en un repositorio varía entre 3% y 11%. Este criterio es utilizado para obtener la mayor cantidad de scripts posibles de acuerdo con la realidad de los respositorios.

[5] https://git-scm.com

Información sobre los repositorios, para aplicar los criterios definidos, es obtenida a través del API REST v3 de GitHub[6] y base de datos del proyecto GHTorrent[7]. Alrededor de 20000 repositorios Puppet estuvieron disponibles cuando este trabajo fue ejecutado, de los cuales, 174 repositorios cumplieron los criterios definidos. Los repositorios fueron descargados y utilizados en las fases de extracción de métricas de proceso e identificación de defectos de seguridad de cada script Puppet.

4.2. Extracción de métricas de proceso de scripts de IaC

Esta fase tiene el propósito de extraer las métricas de proceso de cada script del registro de commits Git de cada repositorio Puppet. Las métricas de proceso recolectadas son descritas a continuación.

- **Número de revisiones (NR):** esta métrica representa el número de revisiones (una revisión aborda la corrección de errores, modificaciones o adición de código) realizadas sobre un script de IaC durante el proceso de desarrollo. En Git, una revisión es registrada en un commit, en virtud de ello, el número de revisiones corresponde al número de commits que afectan el script de IaC.
- **Número de autores de revisiones (NAR):** esta métrica representa el número de autores diferentes que revisan el script de IaC durante el proceso de desarrollo. En Git, el número de autores de revisiones corresponde al número de autores de commits que afectan el script de IaC.
- **Número de líneas modificadas (NLM):** esta métrica representa el número total de líneas modificadas en las revisiones realizadas sobre un script de IaC durante el proceso de desarrollo. En Git, el número de líneas modificadas corresponde a la suma de líneas añadidas y líneas eliminadas en los commits que afectan al script de IaC.
- **Número promedio de líneas modificadas por revisión (NPLMR):** esta métrica representa el número promedio de líneas modificadas en cada revisión realizada sobre un script de IaC durante el proceso de desarrollo. En Git, el número promedio de líneas modificadas por revisión representa el número de líneas modificadas en los commits entre el número de commits que afectan al script de IaC.
- **Edad del script (ES):** esta métrica representa el número de días entre la primera revisión y última revisión realizada sobre un script de IaC durante el proceso de

[6] https://developer.github.com/v3/

[7] http://ghtorrent.org/

desarrollo. En Git, la edad del script corresponde al número de días entre el primer commit y el último commit que afecta al script de IaC. En este trabajo, el último commit es el commit disponible cuando la recolección de métricas fue ejecutada.

- **Número promedio de días entre revisiones (NPDR):** esta métrica representa el número promedio de días entre revisiones realizadas sobre un script de IaC durante el proceso de desarrollo. En Git, el número promedio de días entre revisiones corresponde al número de días de la edad del script entre el número de commits que afectan al script de IaC.

Las métricas de proceso de scripts Puppet fueron extraídas usando comandos de Git presentes en el script disponible en el Anexo I.

4.3. Identificación de defectos de seguridad en scripts de IaC

El propósito de esta fase es identificar defectos de seguridad de cada script Puppet, usando análisis estático de código. En este trabajo, los smells de seguridad identificados en el estudio de (Rahman, Parnin, & Williams, 2019), presentados en la Tabla 2, son considerados defectos de seguridad. En aquel estudio, un smell de seguridad es definido como un patrón de código que señala debilidad en la seguridad y puede conducir a brechas de seguridad; esta definición de smell está acorde con la definición de defecto de software establecida por (IEEE, 2010), el cual define un defecto como una imperfección que necesita ser reparada o reemplazada.

Tabla 2 - Defectos de seguridad que pueden ser encontrados en scripts Puppet

DEFECTO DE SEGURIDAD	DESCRIPCIÓN
Administrador por defecto	Asignar usuarios como usuario administrador. Este defecto implica no cumplir principio de menor privilegio.
Contraseña vacía	Asignar cadena de caracteres de longitud 0 a un atributo o variable relacionado con una contraseña.
Secretos codificados	Escribir información confidencial como nombre de usuarios, contraseñas, claves criptográficas, en el script.
Comentarios sospechosos	Escribir información sobre defectos, debilidades, funcionalidad incompleta, en comentarios.
Acceso sin restricción	Asignar valor de dirección IP 0.0.0.0 para bases de datos o instancias de nube.
HTTP sin TLS	Usar HTTP (Hypertext Transfer Protocol) sin TLS (Transport Layer Security)
Algoritmo de criptografía débil	Usar algoritmos criptográficos débiles, como MD5 (Message-Digest Algorithm) y SHA1 (Secure Hash Algorithm).

Fuente: (Rahman, Parnin, & Williams, 2019)

Los defectos de seguridad de la Tabla 2 están relacionados con la asignación de valor a un atributo o variable, tal y como puede ser observado en el fragmento de código Puppet de la
Figura 5.

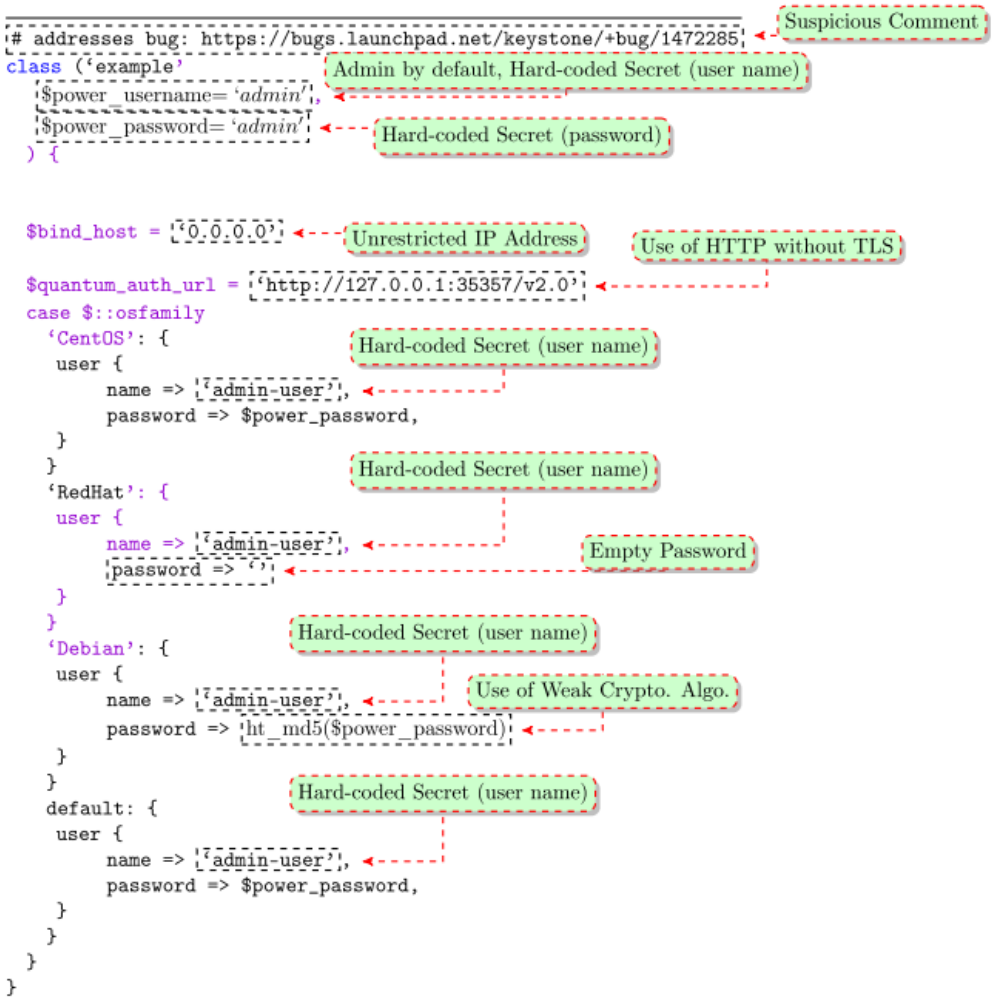

Figura 5 - Fragmento de script Puppet con defectos de seguridad (Rahman, Rahman, Parnin, & Williams, 2019)

Los defectos de seguridad listados en la Tabla 2 pueden ser identificados mediante la herramienta de análisis estático de código SLIC (Rahman A. , 2019). SLIC fue creada por (Rahman, Parnin, & Williams, 2019), está compuesta por tres motores de análisis estático para identificar defectos de seguridad en tres tipos de scripts de IaC (Puppet, Chef y Ansible) y está disponible bajo la licencia MIT[8]. El motor de análisis estático para Puppet

[8] https://opensource.org/licenses/MIT

es un módulo chequeador de Puppet-Lint[9] que puede funcionar de forma independiente, y es utilizado en este trabajo como herramienta de análisis estático de código para identificar los defectos de seguridad en los scripts Puppet recolectados. El código del chequeador Puppet-Lint está disponible en el Anexo II.

La Figura 6 muestra un fragmento de código Puppet con defectos de seguridad HTTP sin TLS y la Figura 7 muestra el resultado que genera la herramienta de análisis estático de código luego de analizar el script que contiene ese fragmento. El resultado lista los defectos de seguridad y el número de línea en donde puede ser encontrado. El número de defectos de seguridad en cada script Puppet es la suma de todos los tipos de defectos identificados con el chequeador de Puppet-Lint.

```
39
40      class { '::neutron::agents::metadata':
41        auth_password => $::openstack::config::neutron_password,
42        shared_secret => $::openstack::config::neutron_shared_secret,
43        auth_url      => "http://${controller_management_address}:35357/v2.0",
44        debug         => $::openstack::config::debug,
45        auth_region   => $::openstack::config::region,
46        metadata_ip   => $controller_management_address,
47        enabled       => true,
48      }
49    }
50
51    anchor { 'neutron_common_first': } ->
52    class { '::neutron::server::notifications':
53      nova_url            => "http://${controller_management_address}:8774/v2/",
54      nova_admin_auth_url => "http://${controller_management_address}:35357/v2.0/",
55      nova_admin_password => $::openstack::config::nova_password,
56      nova_region_name    => $::openstack::config::region,
57  } ->
```

Figura 6 - Código Puppet con defecto de seguridad HTTP sin TLS

En el resultado de la herramienta de análisis estático de código también incluyen alertas sobre las líneas de código que no están de acuerdo con el estilo de código de Puppet. La razón es debido a que el chequeador o herramienta de análisis estático de código identifica los defectos de seguridad además de los defectos definidos por Puppet-Lint.

```
WARNING: arrow should be on the right operand's line on line 30
WARNING: arrow should be on the right operand's line on line 51
WARNING: arrow should be on the right operand's line on line 57
ERROR: openstack::profile::neutron::server not in autoload module layout on line 2
WARNING: SECURITY:::HTTP:::Do not use HTTP without TLS. This may cause a man in the middle attack. Use TLS with
 HTTP.@http://@ on line 43
WARNING: SECURITY:::HTTP:::Do not use HTTP without TLS. This may cause a man in the middle attack. Use TLS with
 HTTP.@http://@ on line 53
WARNING: SECURITY:::HTTP:::Do not use HTTP without TLS. This may cause a man in the middle attack. Use TLS with
 HTTP.@http://@ on line 54
```

[9] https://puppet-lint.com

Figura 7 - Detección de defecto de seguridad HTTP sin TLS con herramienta de análisis estático de código

4.4. Análisis correlacional entre métricas de proceso y defectos de seguridad

La extracción de métricas de proceso e identificación de defectos de seguridad permiten construir un conjunto de datos sobre métricas de proceso y número de defectos de seguridad presentes en cada script Puppet, el cual está disponible en el Anexo III. La Tabla 3 muestra información descriptiva sobre este conjunto de datos, usando los códigos descritos a continuación. Estos códigos también son usados en la Tabla 6.

- **NR**: Número de revisiones
- **NAR**: Número de autores de revisiones
- **NLM**: Número de líneas modificadas
- **NPLMR**: Número promedio de líneas modificadas por revisión
- **ES**: Edad del script
- **NPDR**: Número promedio de días entre revisiones
- **NDS**: Número de defectos de seguridad

Tabla 3 - Información descriptiva de métricas de proceso y defectos de seguridad en scripts Puppet

	NR	NAR	NLM	NPLMR	ES	NPDR	NDS
Promedio	12	7	192	21	755	90	2
Desviación estándar	20	12	373	33	724	105	5
Mínimo	1	1	0	0	0	0	0
Máximo	548	297	7977	814	4005	922	85

n = 6546

4.4.1. Evaluación de distribución de número de defectos de seguridad

Es necesario evaluar la distribución de las muestras de las variables con el propósito de determinar si son similares a la distribución normal, de tal manera, elegir el método para calcular el coeficiente de correlación entre cada métrica de proceso y defectos de seguridad. Si la distribución de las métricas de proceso y número de defectos de seguridad son semejantes a la distribución normal, entonces la correlación Pearson es utilizada, caso contrario, la correlación Spearman es utilizada. Hay varias técnicas para determinar si la distribución de una variable es similar a la distribución normal, por ejemplo, histograma,

gráfico de cuantiles o Q-Q (NIST/SEMATECH, 2012), prueba de normalidad (Wilk & Shapiro, 1965). En este trabajo, la técnica gráfico de cuantiles es utilizada, debido a que la comparación de distribución permite distinguir con claridad si dos distribuciones son similares.

La Figura 8 muestra el gráfico de cuantiles de la muestra de número de defectos de seguridad comparada con los cuantiles de una distribución normal, en el cual es posible identificar que las medidas de número de defectos de seguridad (color azul), no están distribuidas de forma similar a la distribución normal (línea roja), por lo tanto, el coeficiente de correlación Spearman es utilizado. El gráfico de cuantiles fue generado con el módulo statsmodels[10], y el archivo con las sentencias utilizadas está disponible en el Anexo I.

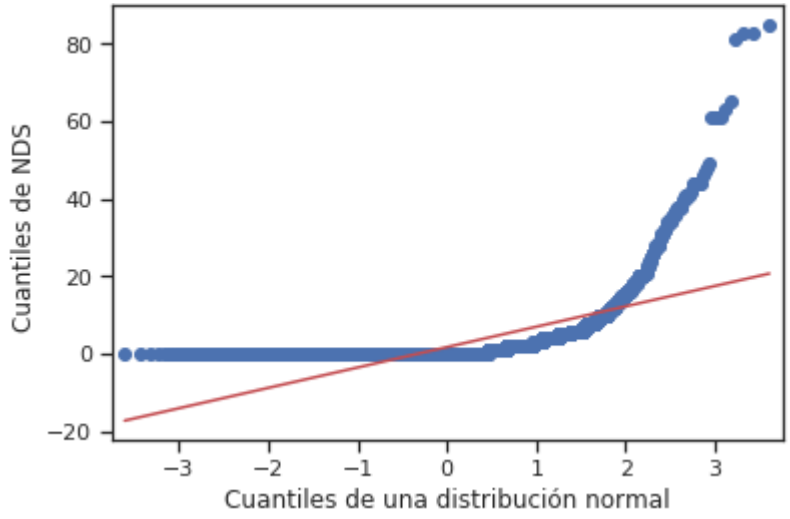

Figura 8 - Gráfico de cuantiles entre la distribución de número de defectos de seguridad y distribución normal

4.4.2. Cálculo de coeficiente de correlación

El siguiente paso en el análisis correlacional es calcular el coeficiente de correlación Spearman entre cada métrica de proceso y el número de defectos de seguridad, para identificar si hay relación entre las variables. Para cada par de variables de la Tabla 3 , por ejemplo, métrica de proceso número de autores de revisión y número de defectos de seguridad, es calculado el coeficiente de correlación; el mismo procedimiento es aplicado con las otras métricas de proceso y el número de defectos de seguridad. Después de

[10] http://www.statsmodels.org/stable/index.html

calcular el coeficiente de correlación, la prueba de hipótesis de correlación es ejecutada con nivel alfa de 0.05 para evaluar si la correlación calculada usando una muestra es representativa de la población. El análisis correlacional termina con la interpretación del coeficiente de correlación de acuerdo con la información presentada en la Tabla 1.

Los módulos, pandas[11], penguin[12], pyplot[13], son usados para procesar el conjunto de datos, calcular el coeficiente de correlación, p-valor, y el script con las sentencias de estos módulos está disponible en el Anexo I. Por ejemplo, la Figura 9 muestra las líneas para desplegar el gráfico de dispersión de la Figura 10 y calcular el coeficiente de correlación entre el número de revisiones y número de defectos de seguridad, mostrado en la Figura 11.

```
######## Numero de revisiones y Numero de defectos de seguridad
    # Grafico de dispersion
    plt.scatter(x=scriptsds['NR'],y=scriptsds['NDS'],marker='o', c='b', edgecolor='k')
    plt.xlabel('Número de revisiones')
    plt.ylabel('Número de defectos de seguridad')
    plt.show()
    # Coeficiente de correlacion
    coef_NR_NDS = pg.corr(x=scriptsds['NR'], y=scriptsds['NDS'], method='spearman')
    print("\n Coeficiente de correlacion Spearman entre numero de revisiones \
y numero de defectos de seguridad: \n", coef_NR_NDS)
```

Figura 9 - Código para generar un gráfico de dispersión y calcular el coeficiente de correlación, p-valor

Es recomendado analizar el gráfico de dispersión entre las variables para identificar posibles valores atípicos que pueden señalar una relación entre pocas medidas, lo cual puede incrementar el valor del coeficiente de correlación, señalando que existe relación entre las variables cuando en realidad no es así. En este caso, el método Spearman es resistente a valores atípicos, por lo tanto, los resultados de coeficiente de correlación están acorde con la asociación lineal que existe entre cada métrica de proceso y defectos de seguridad.

[11] https://pandas.pydata.org/

[12] https://pypi.org/project/pingouin/

[13] https://matplotlib.org/index.html

La Figura 10 presenta un gráfico de dispersión entre el número de revisiones y número de defectos de seguridad, el cual permite identificar pocos valores extremos, y en la mayoría de los puntos no es posible visualizar asociación lineal entre las variables. Esta falta de asociación entre las variables puede ser comprobado con el coeficiente de correlación Spearman de 0.15, registrado en la Tabla 6, y el cual de acuerdo con la Tabla 1, señala una relación muy débil entre estas variables.

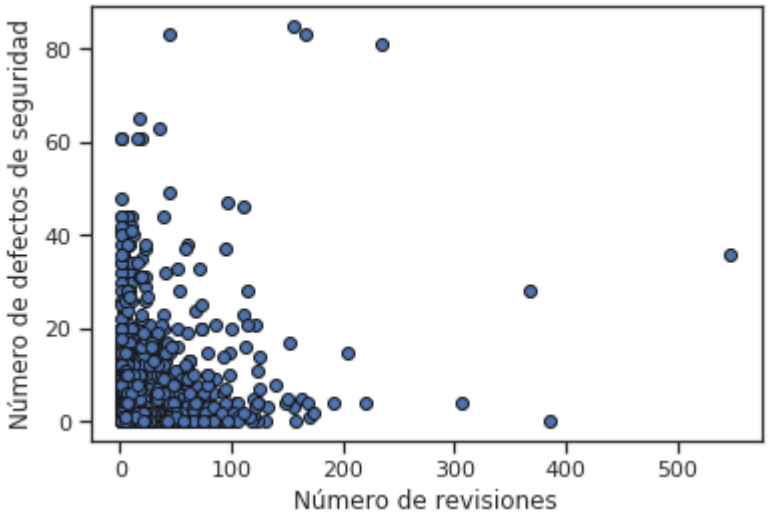

Figura 10 - Número de revisiones versus número de defectos de seguridad

Coeficiente de correlacion Spearman entre numero de revisiones y numero de defectos de seguridad:

	n	r	CI95%	r2	adj_r2	p-val	power
spearman	6545	0.155	[0.13, 0.18]	0.024	0.024	2.204773e-36	1.0

Figura 11 - Coeficiente de correlación entre número de revisiones y número de defectos de seguridad con p-valor.

La matriz de correlación de la Tabla 6, y los gráficos de dispersión de la Figura 12, Figura 13, Figura 14, Figura 15, Figura 16, Figura 17, muestran los resultados de esta fase.

5. RESULTADOS Y DISCUSIÓN

Esta sección presenta e interpreta los resultados obtenidos luego de ejecutar la metodología detallada en el capítulo anterior.

5.1. Conjunto de datos de repositorios de scripts de IaC

Alrededor de 20000 repositorios de lenguaje Puppet estaban disponibles en GitHub cuando el presente estudio fue ejecutado, de los cuales, 174 repositorios cumplieron los criterios definidos en la sección 4.1, y fueron descargados para extraer las medidas de las métricas de proceso y el número total de defectos de seguridad presentes de cada script Puppet. Información descriptiva sobre los repositorios descargados es presentada en la Tabla 4. El conjunto de datos de repositorios puede ser encontrado en el Anexo IV.

Tabla 4 - Información descriptiva sobre repositorios de scripts Puppet

	SCRIPTS PUPPET	**ARCHIVOS NO IaC**	**REVI-SIONES**	**AUTORES DE REV.**	**EDAD (DÍAS)**	**REV. POR MES**
Promedio	38	78	652	48	1709	12
Desviación estándar	78	216	2224	55	765	25
Min.	0	1	54	10	126	2
Max.	776	1892	28047	388	4005	292

El número promedio de archivos no IaC es superior al número promedio de scripts Puppet. Este hallazgo no fue esperado, ya que, en GitHub, el lenguaje de un repositorio es asignado por el lenguaje de programación con el cual la mayoría de los archivos son escritos. En este trabajo, solo repositorios de lenguaje Puppet fueron considerados y la expectativa fue que la mayoría de los archivos presentes en un repositorio deberían ser scripts Puppet.

5.2. Defectos de seguridad identificados

El número total de scripts Puppet analizados con la herramienta de análisis estático fue 6545, de los cuales, 2123 scripts presentan defectos de seguridad. La Tabla 5 muestra detalle de los defectos de seguridad identificados. El defecto de seguridad más común es *secretos codificados* con 9395 ocurrencias, afectando a 1585 scripts Puppet, lo cual corresponde al 75% de los scripts recolectados.

Tabla 5 - Ocurrencias de defectos de seguridad en scripts Puppet

DEFECTO DE SEGURIDAD	OCURRENCIAS	SCRIPTS AFECTADOS
Administrador por defecto	52	50
Algoritmo de criptografía débil	106	61
Acceso sin restricción	219	107
Contraseña vacía	271	127
Comentarios sospechosos	661	418
HTTP sin TLS	836	422
Secretos codificados	9395	1585
Defecto de seguridad general (todos los tipos)	11540	2123

5.3. Correlación entre métricas y defectos de seguridad

El coeficiente de correlación entre las métricas de proceso y defectos de seguridad, calculado con el método Spearman es mostrado en la matriz de correlación de la Tabla 6. De acuerdo con los resultados del p-valor, el coeficiente de correlación Spearman entre casi todas las métricas de proceso y el número de defectos de seguridad en scripts Puppet es representativo (p-valor<0.05), con la excepción presente en la correlación entre el número promedio de días entre revisiones y número de defectos de seguridad, ya que el p-valor es mayor que el nivel alfa. De acuerdo con la interpretación defina en la Tabla 1, los coeficientes de correlación de la Tabla 6 señalan que el nivel de correlación entre todas las métricas de proceso y el número de defectos de seguridad es muy débil.

Tabla 6 - Matrix de correlación Spearman de métricas de procesos y defectos de seguridad de scripts Puppet

	NR	NAR	NLM	NPLMR	ES	NPDR
NDS	+.15	+.13	+.23	+.13	+.10	-.01*
NR		+.88	+.80	-.04	+.78	+.27
NAR			+.75	+.03	+.76	+.32
NLM				+.53	+.57	+.12
NPLMR					-.10	-.12
ES						+.71

n = 6734
p<0.05
*p>0.05

En la matriz de correlación también es presentado el coeficiente de correlación entre métricas de proceso, lo cual permite identificar relaciones entre estas, por ejemplo, el

número de revisiones y el número de autores de revisiones tienen asociación lineal fuerte, el número de revisiones y el número de líneas modificadas tienen asociación lineal fuerte, el número de autores de revisiones y el número de líneas modificadas tienen asociación lineal fuerte. Estas asociaciones son esperadas, por su naturaleza; si el número de autores de revisiones incrementa, es esperado que el número de revisiones y número de líneas modificadas también incrementa.

La Figura 12 muestra la distribución de medidas de número de revisiones versus número de defectos de seguridad. La concentración de medidas está presente en el rango de 0 a 100 del eje número de revisiones y en el rango de 0 a 40 del eje de número de defectos de seguridad. Varios valores atípicos están presentes, los cuales pueden influenciar en el cálculo del coeficiente de correlación, como fue mencionado en el análisis correlacional. Sin embargo, en la gráfica de dispersión no existe relación entre las variables, lo cual es confirmado con el coeficiente de correlación calculado.

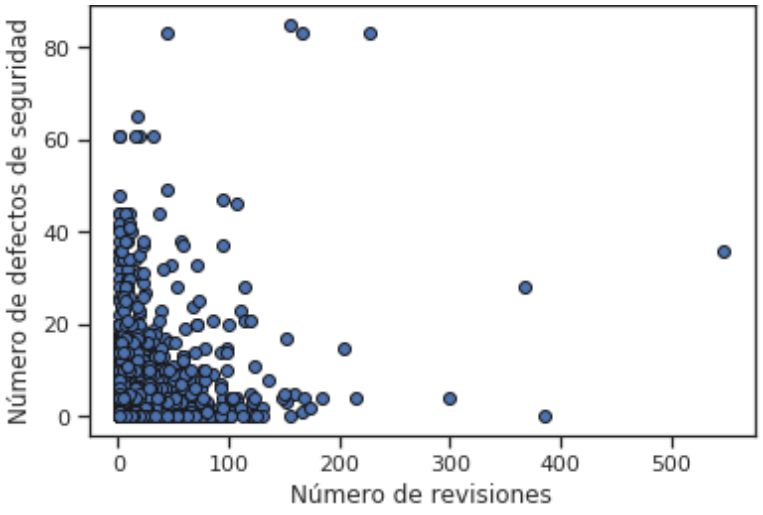

Figura 12 - Número de revisiones y número de defectos de seguridad

La Figura 13 muestra la distribución de medidas de número de autores de revisiones versus número de defectos de seguridad. La concentración de medidas está presente en el rango de 0 a 50 del eje número de autores de revisiones y en el rango de 0 a 20 del eje de número de defectos de seguridad. De igual manera, que en el grafico anterior, no hay asociación lineal, y es confirmado con el coeficiente de correlación.

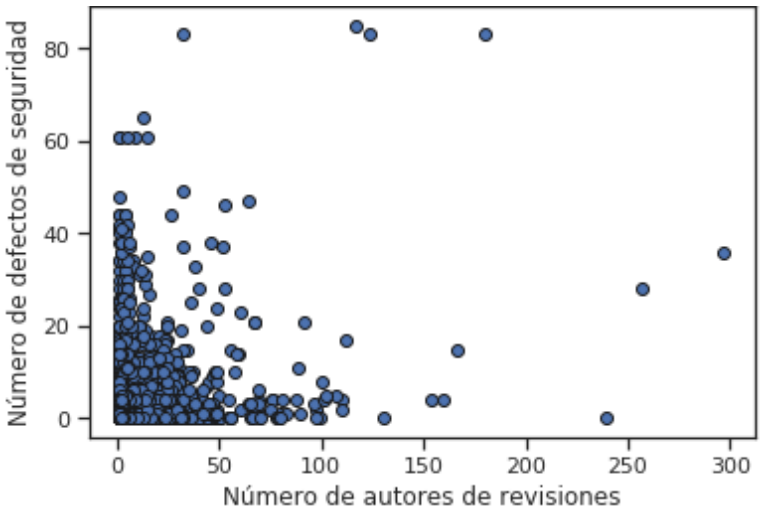

Figura 13 - Número de autores de revisiones y número de defectos de seguridad

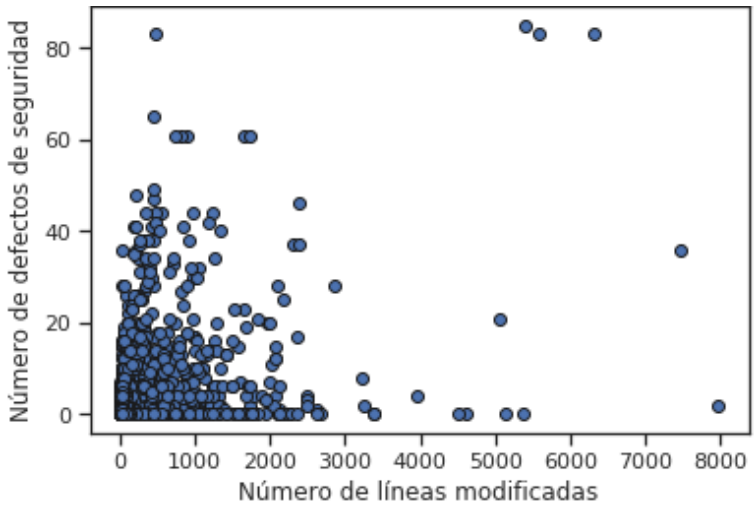

Figura 14 - Número de líneas modificadas y número de defectos de seguridad

La Figura 14 muestra la distribución de medidas de número de líneas modificadas versus número de defectos de seguridad. Las medidas están concentradas en el rango de 0 a 2000 en el eje número de líneas modificadas y en el rango de 0 a 30, aproximadamente, del eje de número de defectos de seguridad. En el gráfico de dispersión es posible distinguir mayor cantidad de medidas dispersas, lo cual puede generar confusión y señalar una relación que no existe. Es así, que en el coeficiente de correlación entre el número de líneas modificadas y número de defectos de seguridad es 0.23, el coeficiente de mayor magnitud respecto a los otros pares de variables, sin embargo, la interpretación del coeficiente permite determinar que la consistencia de la relación es débil, y en el gráfico este hallazgo es comprobado.

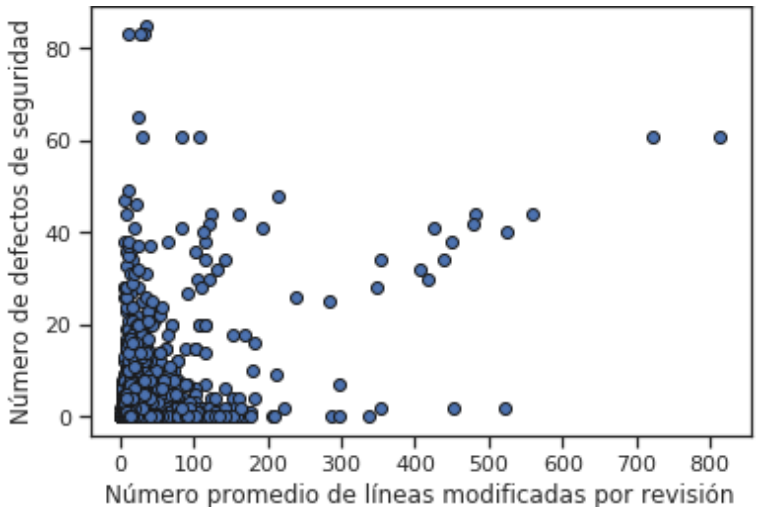

Figura 15 - Número promedio de líneas modificadas y número de defectos de seguridad

La Figura 15 muestra la distribución de medidas de número promedio de líneas modificadas por revisión versus número de defectos de seguridad. Las medidas están dispersas en un rango amplio, tanto en el eje número promedio de líneas modificadas, como en el eje número de defectos de seguridad, indicando que la consistencia de la correlación es muy débil. Esto puede ser validado con el coeficiente de correlación de 0.13 registrado en la Tabla 6.

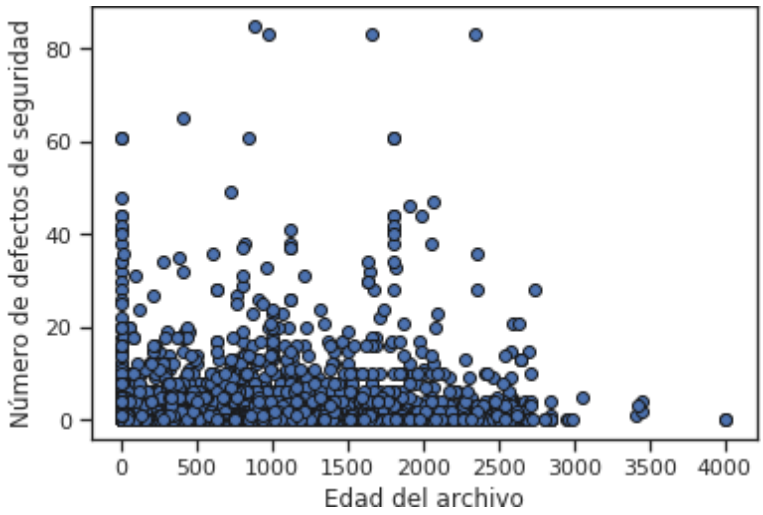

Figura 16 - Edad del archivo y número de defectos de seguridad

La Figura 16 muestra la distribución de medidas de edad del archivo o script versus número de defectos de seguridad. Las medidas están dispersas en un rango amplio, tanto en el eje edad del script, como en el eje número de defectos de seguridad, señalando una relación muy débil entre las variables

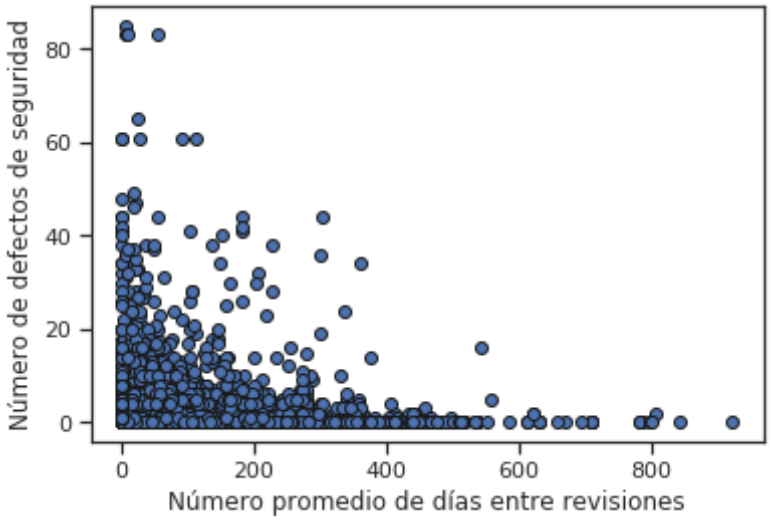

Figura 17 - Número promedio de días entre revisiones y número de defectos de seguridad

La Figura 17 muestra la distribución de medidas de número promedio de días entre revisiones versus número de defectos de seguridad. Las medidas están dispersas en un rango amplio, tanto en el eje número promedio de días entre revisiones, como en el eje número de defectos de seguridad. Es así, que el coeficiente de correlación Spearman entre número promedio de días entre revisiones y número de defectos de seguridad es casi 0 (ver Tabla 6).

5.4. Discusión

Los scripts de IaC presentan debilidades de seguridad, identificadas en el trabajo de (Rahman, Parnin, & Williams, 2019), en el cual determinaron que no todos presentan este tipo de defecto. Este hallazgo despierta el interés por determinar los factores en común en los scripts de IaC con este tipo de defectos, los cuales pueden ser utilizados en actividades de aseguramiento de calidad de código. Por ejemplo, identificar aquellos scripts más propensos a tener estas debilidades mediante los factores determinados, construir modelos de predicción de scripts de IaC con defectos en base a los factores asociados a ese tipo de defecto. El presente trabajo contribuye al estado del arte con el análisis correlacional entre métricas de proceso y defectos de seguridad en scripts Puppet, para determinar el nivel de asociación lineal entre estas dos variables, y de esta manera conocer si algunas métricas de proceso son factores en común en aquellos scripts Puppet defectuosos.

De acuerdo con los resultados encontrados y presentados en la sección anterior (ver Tabla 6) la asociación lineal o correlación entre las métricas de proceso número de revisiones,

número de autores de revisiones, edad del script, número promedio de días entre revisiones, número de líneas modificadas, número promedio de líneas modificadas por revisión y el número de defectos de seguridad es muy débil. El coeficiente de correlación es menor o igual que 0.23, lo cual significa que en menos del 5.2 % de los scripts Puppet, el número de defectos seguridad y las métricas antes mencionadas, presentan asociación o varían en conjunto. Estos resultados permiten determinar que las métricas de proceso, antes mencionadas, no son factores en común en los scripts Puppet con defectos de seguridad.

Las métricas de proceso consideradas en este trabajo fueron seleccionadas en base a los resultados de estudios previos sobre métricas de proceso correlacionadas con archivos de código fuente con defectos (Meneely & Williams, 2009), (Misirli, Murphy, Zimmermann, & Basar, 2011), (Illes-Seifert & Barbara, 2010), (Madeyski & Jureczko, 2015), (Rahman & Williams, 2019). En estos estudios, los defectos analizados están relacionados con varias propiedades del archivo o código, como, complejidad, estructura, asignación de variables, etc. (Rahman & Williams, 2019) estudió scripts de IaC, pero considerando defectos similares a los defectos de los estudios de archivos de código fuente. En base a estos hechos, es posible identificar una razón por la cual las métricas de procesos presentan correlación muy débil o despreciable con el número de defectos de seguridad en scripts de IaC, la cual es la diferencia entre los defectos analizados en cada trabajo. Los defectos objetivo en el presente trabajo son defectos de seguridad, relacionados principalmente con la asignación de estado a atributos Puppet, similar a la asignación de variables, lo cual es una particularidad de los defectos considerados en estudios previos de archivos de código fuente y scripts de IaC.

Por otro lado, los repositorios de scripts Puppet recopilados en este trabajo académico fueron obtenidos aplicando criterios para filtrar repositorios que no aportan información de proyectos de Ingeniería de Software promedio. Los criterios fueron definidos en base a estudios previos (Meneely & Williams, 2009), (Jiang & Adams, 2015), (Munaiah, Kroh, Craig, & Nagappan, 2017). De acuerdo con (Munaiah, Kroh, Craig, & Nagappan, 2017), un repositorio de código proporciona información sobre el proceso de desarrollo, de tal manera, es factible considerar que la asociación lineal entre métricas de proceso (identificadas como métricas asociadas con defectos en archivos de código fuente), y defectos de seguridad en scripts Puppet es muy débil, debido a que el desarrollo de scripts de IaC y proceso de desarrollo de software convencional es diferente. Esta hipótesis puede ser fundamentada en los resultados obtenidos el estudio de (Jiang & Adams, 2015), en el

cual identificaron que los scripts Puppet son editados con mayor frecuencia y son de mayor tamaño que un archivo de código fuente o archivo de construcción.

En base a estos resultados, es posible argumentar que los scripts Puppet no son desarrollados de la misma forma que otro artefacto de software, y es otra razón por la cual las métricas de proceso y defectos de seguridad en scripts Puppet presentan asociación lineal muy débil que puede ser considerada despreciable.

6. CONCLUSIONES

Los scripts de IaC son una herramienta fundamental en las actividades de despliegue continuo de software y gestión de infraestructura, por lo tanto, es importante identificar y solucionar defectos que afectan su mantenibilidad y seguridad. Trabajos previos en el dominio de IaC estudiaron debilidades de seguridad en scripts Puppet, y determinaron que no todos presentan estos problemas. De igual manera, estudios previos en Ingeniera de Software identificaron que ciertas métricas de procesos son factores relacionados a los defectos presentes en archivos de código fuente.

Este trabajo fue fundamentado en estos estudios, y ejecutó el análisis correlacional entre métricas de proceso, como, número de revisiones, número de autores de revisiones, número de líneas modificadas, número promedio de líneas modificadas por revisión, edad del script o archivo, número promedio de días entre revisiones y el número de defectos de seguridad en scripts Puppet, para determinar el grado de relación que existe entre estas variables, y de tal manera, conocer sin son factores en común en scripts Puppet con defectos de seguridad. Los scripts y métricas de proceso fueron obtenidos de repositorios Puppet de GitHub. Las métricas fueron extraídas del historial de commits de Git, y los defectos fueron identificados mediante con la herramienta, de análisis estático de código, SLIC. La correlación fue calculada usando el método Spearman, debido a que la muestra de número de defectos de seguridad en scripts Puppet no es similar a la distribución normal.

El desarrollo de este proyecto permitió construir conjuntos de datos sobre métricas de procesos y defectos de seguridad en scripts Puppet, y repositorios Puppet. Los resultados del análisis correlacional permitieron determinar que las métricas de procesos no son factores relacionados con defectos de seguridad en scripts Puppet debido a que cada métrica de proceso y el número de defectos de seguridad presentan correlación muy débil, con coeficiente de correlación Spearman inferior o igual a 0.23. Es recomendado estudiar diferentes características o proceso de desarrollo, para encontrar factores relacionados con defectos de seguridad en scripts Puppet u otras tecnologías de IaC, como Ansible, Chef, entre otras.

REFERENCIAS BIBLIOGRÁFICAS

Aggarwal, R., & Ranganathan, P. (2016). Common pitfalls in statistical analysis: The use of correlation techniques. *Perspectives in clinical research*, (págs. 187–190).

Akond, R. (2019). *Security Linter for Infrastructure as Code (SLIC)*. Obtenido de https://github.com/akondrahman/IacSec

Boehm, B., & Basili, V. (2001). Software Defect Reduction Top 10 List. *Computer*, 135-137.

Frederick, G., & Larry, W. (2013). *Statistics for the behavioral sciences 9th Edition.* Belmont: Wadsworth Cengage Learning.

GitHub, I. (10 de septiembre de 2019). *GitHub glossary*. Obtenido de https://help.github.com/en/articles/github-glossary

Guckenheimer, S. (2017). *What is Infrastructure as Code?* Obtenido de https://docs.microsoft.com/en-us/azure/devops/learn/what-is-infrastructure-as-code

Haldun, A. (2018). User's guide to correlation coefficients. *Turkish journal of emergency medicine*, (págs. 91-93).

Hofacker, A. (2008). *Rapid lean construction - quality rating model.* Manchester: s.n.

Hryszko, J., Madeyski, L., Dąbrowska, M., & Konopka, P. (2017). Defect prediction with bad smells in code. *arXiv:1703.06300 [cs.SE].*

IEEE. (2010). *Ieee standard classification for software anomalies, IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993).*

Illes-Seifert, T., & Barbara, P. (2010). Exploring the relationship of a file's history and its fault-proneness: An empirical method and its application to open source programs. *Information and Software Technology*, 539-558.

IRI. (10 de octubre de 2019). *Statistical Techniques in the Data Library: A Tutorial*. Obtenido de http://iridl.ldeo.columbia.edu/dochelp/StatTutorial/

ITRC. (1 de diciembre de 2013). *Groundwater Statistics and Monitoring Compliance, Statistical Tools for the Project Life Cycle*. Obtenido de http://www.itrcweb.org/gsmc-1/

Jiang, Y., & Adams, B. (2015). Co-evolution of Infrastructure and Source Code - An Empirical Study. *2015 12th Working Conference on Mining Software Repositories*, (págs. 45-55).

Kan, S. (1995). *Metrics and Models in Software Quality Engineering.* Sin ciudad: Sin editorial.

Kanies, L. (2 de 8 de 2012). *Why Puppet has its own configuration language*. Obtenido de https://puppet.com/blog/why-puppet-has-its-own-configuration-language

Koskela, L. (1992). *Application of the new production philosophy to construction.* Finland: VTT Building Technology.

Madeyski, L., & Jureczko, M. (2015). Which process metrics can significantly improve defect prediction models? An empirical study. *Software Quality Journal*, 393-422.

Meneely, A., & Williams, L. (2009). Secure open source collaboration: an empirical study of linus' law. *CCS '09 Proceedings of the 16th ACM conference on Computer and communications securit*, (págs. 453-462).

Meneely, A., & Williams, L. (2009). Secure open source collaboration: an empirical study of linus' law. *Proceedings of the 16th ACM conference on Computer and communications security* , (págs. 453-462). Chicago, USA.

Mindrila, D., & Balentyne, P. (31 de octubre de 2019). *Scatterplots and Correlation.* Obtenido de https://www.westga.edu/academics/research/vrc/assets/docs/scatterplots_and_correlation_notes.pdf

Misirli, A., Murphy, B., Zimmermann, T., & Basar, A. (2011). An Explanatory Analysis on Eclipse Beta-Release Bugs Through In-Process Metrics. *Proceedings of the 8th International Workshop on Software Quality (WoSQ 2011)*, (págs. 26-33).

Mitchell, A. (21 de agosto de 2014). *Getting Started With Puppet Code: Manifests and Modules*. Obtenido de https://www.digitalocean.com/community/tutorials/getting-started-with-puppet-code-manifests-and-modules

Moore, D., Notz, W., & Flinger, M. (2013). *The basic practice of statistics 6th edition.* New York: W. H. Freeman and Company.

Mukaka, M. (2012). Statistics corner: A guide to appropriate use of correlation coefficient in medical research. *The journal of medical association of malawi*, (págs. 69-71).

Munaiah, N., Kroh, S., Craig, C., & Nagappan, M. (2017). Curating GitHub for engineered software projects. *Empirical Software Engineering*, 3219-3253.

NIST/SEMATECH. (April de 2012). *e-Handbook of Statistical Methods*. Obtenido de http://www.itl.nist.gov/div898/handbook/

Nord, R., Ozkaya, I., Schwartz, E., Shull, F., & Kazman, R. (2016). Can Knowledge of Technical Debt Help Identify Software Vulnerabilities? *9th Workshop on Cyber Security Experimentation and Test (CSET 16).* Austin, Texas: Unisex Association.

Ogee, A., Ellis, M., Scibilia, B., Pammer, C., & Steele, C. (19 de 03 de 2015). *Understanding Hypothesis Tests: Significance Levels (Alpha) and P values in Statistics*. Obtenido

de https://blog.minitab.com/blog/adventures-in-statistics-2/understanding-hypothesis-tests-significance-levels-alpha-and-p-values-in-statistics

Price, P., Jhangiani, R., & I-Chant. (20 de 11 de 2019). *Correlational Research*. Obtenido de https://opentextbc.ca/researchmethods/chapter/correlational-research/

Puppet. (10 de Septiembre de 2019). *Language: Basics*. Obtenido de https://puppet.com/docs/puppet/5.3/lang_summary.html

Puppetforge. (15 de noviembre de 2019). *puppetlabs/postgresql*. Obtenido de https://forge.puppet.com/puppetlabs/postgresql

Rahman, A., & Williams, L. (2018). Characterizing Defective Configuration Scripts Used for Continuous Deployment. *IEEE 11th International Conference on Software Testing, Verification and Validation*, (págs. 34-45).

Rahman, A., & Williams, L. (2019). Source code properties of defective infrastructure as code scripts. *Information and Software Technology*, 148-163.

Rahman, A., Parnin, C., & Williams, L. (2019). The seven sins security smells in infrastructure as code scripts. *ProceedingICSE '19 Proceedings of the 41st International Conference on Software Engineering*, (págs. 164-175). Montreal.

Rahman, A., Rahman, M., Parnin, C., & Williams, L. (2019). Security Smells in Infrastructure as Code Scripts.

Rahman, Mahdavi-Hezaveh, & Williams. (2018). A systematic mapping study of infrastructure as code research. *Information and Software Technology*, 65-77.

Rhaman, A. (11 de septiembre de 2019). Obtenido de Container for SLIC that identifies security smells for Puppet scripts: https://cloud.docker.com/repository/docker/akondrahman/ruby_for_sp/general

Rummel, R. (1976). *Understanding Correlation.* Honolulo: Department of political Science University of Huwaii.

Schwarz, J., Steffens, A., & Lichter, H. (2018). Code Smells in Infrastructure as Code. *Quality of Information and Communications Technology.* Coimbra - Portugal.

Sharma, T., Fragkoulis, M., & Spinellis, D. (2016). Does your configuration code smell? *Proceedings of the 13th International Conference on Mining Software Repositories*, (págs. 189-200). Ausitn, Texas.

Sommerville, I. (2011). *Software Enginering.* Boston-USA: Addison-Wesley.

Stecklein, J. M., Dabney, J., Dick, B., Haskins, B., Lovell, R., & Moroney, G. (2004). Error Cost Escalation Through the Project Life Cycle. *14th:; Annual International Symposium, International Council on Systems Engineering.* Toulousen - France .

Wilk, M., & Shapiro, S. (1965). An analysis of variance test for normality (complete samples). *Biometrika*, 591–611.

ANEXOS

Los Anexos descritos a continuación están disponibles en el CD adjunto.

Anexo I - Código Python para extraer métricas de procesos, identificar defectos de seguridad y analizar correlación.

Anexo II - Código de herramienta de análisis estático de código.

Anexo III - Conjunto de datos de métricas de proceso y defectos de seguridad.

Anexo IV - Conjunto de datos de repositorios de scripts Puppet.