Previsão de Vazamento de Recursos em Aplicações Android usando Aprendizado de Máquina

Josias Gomes Lima[1], **Rafael Giusti**[1], **Arilo Claudio Dias-Neto**[1]

[1]Instituto de Computação – Universidade Federal do Amazonas (UFAM)
Amazonas – AM – Brasil

{josias,rgiusti,arilo}@icomp.ufam.edu.br

Abstract. *Context: Android mobile applications (apps) handle many resources built into the device, such as camera, media player, and sensors. Problem: When apps acquire these resources without releasing them properly and in a timely manner, an error called a resource leak occurs. This type of error can cause serious problems, such as performance degradation or system failure. Proposal: This work proposes a machine learning-based solution for the prediction of resource leakage in components based. To that end, the DroidLeaks database was used, which contains 292 resource leaks identified in 32 open source and large-scale apps.*

Resumo. *Contexto: As aplicações móveis (apps) na plataforma Android manipulam muitos recursos incorporados no dispositivo, como câmera, reprodutor de mídia e sensores. Problema: Quando as apps adquirem esses recursos sem liberá-los de maneira adequada e em tempo hábil, acontece o erro chamado vazamento de recursos. Esse tipo de erro pode causar problemas sérios, como degradação de desempenho ou falha do sistema. Proposta: Este trabalho propõe uma solução para a predição de vazamento de recursos em componentes baseando-se em aprendizado de máquina. Para isso, utilizou-se a base de dados DroidLeaks que contém 292 vazamentos de recursos identificados em 32 apps de código aberto e de larga escala.*

1. Introdução

Existem mais de 2,7 bilhões de dispositivos que utilizam a plataforma Android [Statista 2020a] e 2,96 milhões de aplicações móveis (simplesmente, apps) disponíveis para download na loja *Google Play* [Statista 2020b]. Os recursos desses dispositivos e a complexidade das apps continuam crescendo rapidamente. Esse crescimento apresenta inúmeros desafios para correção e bom desempenho da app, pois, além dos defeitos tradicionais, há os defeitos relacionados à utilização dos recursos limitados dos dispositivos. Alguns exemplos desses recursos são reprodutor de mídia, memória, câmera e sensores.

Quando uma app utiliza algum desses recursos de forma incorreta, por exemplo, não liberando o recurso após a sua utilização, ocasiona um erro chamado **vazamento de recursos**. Esse erro pode levar à degradação do desempenho, ocasionando, por exemplo, um enorme consumo de energia ou memória e até mesmo falha do sistema [Bhatt and Furia 2020]. Sendo assim, a identificação desses vazamentos, e sua posterior correção, possui grande importância dentro da etapa de verificação e validação durante o desenvolvimento de uma aplicação, visando minimizar este erro. Alguns métodos, como

analisadores estáticos ([Jiang et al. 2017]) e técnicas de teste ([Qian and Zhou 2016]) são encontrados na literatura técnica com o objetivo de auxiliar os desenvolvedores a identificar vazamentos de recursos em suas apps.

Esta pesquisa visa o desenvolvimento de uma solução para a identificação de vazamento de recursos em apps Android utilizando técnicas de aprendizado de máquina. Aprendizado de máquina (AM) é uma subárea de inteligência artificial (IA) que utiliza dados de entrada para alcançar uma tarefa desejada, ainda que não seja estritamente programada para produzir um resultado específico [El Naqa and Murphy 2015]. Diferentes técnicas de AM são utilizadas para detectar padrões existentes nos dados e classificá-los. Essa classificação é entendida como o processo de atribuir a um determinado dado, o rótulo da classe à qual ele pertence. Na tarefa de predição de erro, esse rótulo da classe poderia ser de erro ou sem erro. [Qian and Zhou 2016] utilizam aprendizado de máquina para priorizar casos de teste com maior propensão a revelar vazamentos de recursos, ao passo que nesta pesquisa, propõe-se identificar vazamento de recursos em apps Android.

Para tanto, utilizou-se a base de dados DroidLeaks [Liu et al. 2019] que contém 292 vazamentos de recursos relacionados a 33 classes de recursos diferentes, 11 das quais são específicas da plataforma Android. Até onde se sabe nenhum dos trabalhos correlatos existentes estudou a identificação de vazamentos relacionados a um número tão grande de classes de recursos diversificados [Liu et al. 2019]. O que se busca é a solução de um problema de classificação, pois se dispõe a prever se um componente possui vazamento de recursos ou não. Portanto, estão sendo empregados classificadores de AM comumente utilizados para predição de erro, a saber, *support vector machine (SVM)*, *naive bayes*, *deep neural network (DNN)*, *logistic regression*, *random forest* e *k-nearest neighbor (KNN)* [Efendioglu et al. 2018, Li et al. 2019, Manjula and Florence 2019].

O restante deste artigo está organizado da seguinte forma: a Seção 2 apresenta a fundamentação teórica; a Seção 3, o método de pesquisa e o estado atual do trabalho; a Seção 4 discute os trabalhos relacionados; e, por fim, a Seção 5 apresenta os resultados esperados.

2. Fundamentação teórica

2.1. Teste de Aplicações Móveis

O teste de aplicações móveis se refere aos diferentes tipos de técnicas a serem aplicadas para verificar a funcionalidade, a usabilidade e a consistência da aplicação [Nimbalkar 2013]. Alguns requisitos diferenciam os testes de aplicações móveis de testes de software convencionais, a título de exemplo, as aplicações móveis devem funcionar em plataformas que possuem diferentes sistemas operacionais, recursos de computação, tamanhos de exibição e duração de bateria [Gao et al. 2014].

A técnica de teste predição de defeitos busca identificar os defeitos antes que eles ocorram. Isso permite que ações corretivas possam ser tomadas o quanto antes no desenvolvimento da aplicação. Uma forma de aplicar essa técnica é utilizando aprendizado de máquina, treinando um modelo para predizer se um componente tem defeito ou não, o qual será brevemente explicado na próxima subseção.

2.2. Aprendizado de Máquina

O aprendizado de máquina pode ser amplamente definido como métodos computacionais que usam informações conhecidas para melhorar o desempenho ou para fazer predições precisas [Mohri et al. 2018]. Essas informações podem estar na forma de conjuntos de dados com rótulos (por exemplo, com erro ou sem erro). Em todos os casos, a qualidade e a quantidade das informações são cruciais para o sucesso das predições feitas pelo modelo.

3. Método de pesquisa e estado atual do trabalho

3.1. Método de Pesquisa

A metodologia de pesquisa usada é composta por duas fases: **concepção**, cujo foco é a definição e construção da abordagem; e **avaliação**, focada em analisar a viabilidade da abordagem. Na fase de concepção foi realizado inicialmente um mapeamento sistemático da literatura técnica com o objetivo de identificar, analisar e sintetizar publicações científicas que tratam de identificação de vazamento de recursos em aplicações móveis. Na segunda etapa desta fase foi realizada uma pesquisa de opinião com o objetivo de caracterizar a relevância das categorias de causas de vazamentos identificados no mapeamento sistemático. Na terceira etapa foi feita a definição da abordagem, como o artefato de entrada, as ferramentas a serem utilizadas e os processos utilizados para a identificação dos vazamentos.

Atualmente, foi identificada uma base de dados (DroidLeaks) com vazamentos de recursos em aplicações móveis. No entanto, essa base é pequena e provavelmente será necessário montar uma base de dados maior. Para isso, pode ser seguido o mesmo protocolo utilizado para montar a base de dados *DroidLeaks* em [Liu et al. 2019]. Na fase de avaliação, pretende-se realizar um estudo de caso com aplicações móveis de uma organização da indústria, a fim de avaliar a eficácia (identificação de mais vazamentos com menos falsos alarmes) da proposta em relação ao estado da arte.

3.2. Proposta

A abordagem *LeaksPred* (originada da expressão *Leaks Prediction*, em português, Predição de Vazamentos) foi projetada para identificar vazamentos de recursos em aplicações móveis Android utilizando aprendizado de máquina. Ela possui quatro processos principais: (1) Preparação dos dados, (2) Execução do treinamento, (3) Extração das métricas e (4) Classificação dos componentes. Estes processos são apresentados na Figura 1.

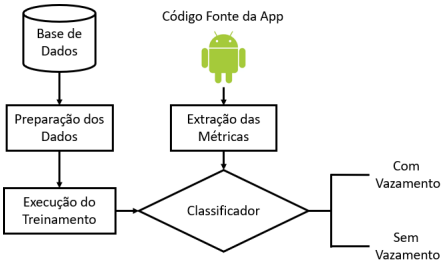

Figura 1. Visão geral da abordagem LeaksPred

Tabela 1. Lista de métricas extraídas

ID	Nome da Métrica	ID	Nome da Métrica	ID	Nome da Métrica	ID	Nome da Métrica
M1	Complexidade Ciclomática	M9	Referências de Variáveis	M17	Halstead Volume	M25	Componentes Externos
M2	Halstead Effort	M10	Componente Cérebro	M18	Halstead Difficulty	M26	Componentes Locais
M3	Profundidade Máxima de Aninhamento	M11	Número de Linhas de Comentário	M19	Profundidade Total de Aninhamento	M27	Variáveis de Instância Referenciadas
M4	Custo de Manutenção Futura	M12	Número de Comentários	M20	Número de Conversões	M28	Exceções Lançadas
M5	Linhas de Código	M13	Número de Declarações	M21	Número de Laços	M29	Modificadores
M6	Número de Parâmetros	M14	Número de Expressões	M22	Número de Operadores	M30	Exceções Referenciadas
M7	Halstead Bugs	M15	Halstead Length	M23	Número de Operandos		
M8	Declarações de Variáveis	M16	Halstead Vocabulary	M24	Referências de Classe		

A seguir serão detalhadas cada etapa que compõe os processos apresentados na Figura 1.

3.2.1. Preparação dos Dados

Foi realizada uma busca na literatura técnica e encontrou-se a base de dados *DroidLeaks* [Liu et al. 2019], que possui 292 componentes com vazamentos de recursos identificados em 32 aplicações Android de código aberto e de larga escala. Para a caracterização desses componentes, foram escolhidas as métricas apresentadas na Tabela 1, a maioria dessas métricas é comumente utilizada para a predição de erro em componentes [Efendioglu et al. 2018, Li et al. 2019, Manjula and Florence 2019]. A ferramenta JHawk[1] será utilizada para ajudar na extração dessas métricas. Ainda não existe um estudo que mostre a relação das métricas da Tabela 1 com vazamentos de recursos. Por isso, pretende-se analisar essa relação, além de agregar outras métricas que se mostrem relacionadas.

Após a extração das métricas para todos os componentes de cada classe[2] que contenha ao menos um componente com vazamento de recursos, será rotulado cada componente no conjunto de dados da abordagem como tendo vazamento de recurso ou não, conforme a Equação 1.

$$Componente(x) = \begin{cases} 1, \text{ se } x \text{ contém vazamento de recurso (erro)} \\ 0, \text{ caso contrário} \end{cases} \quad (1)$$

Com a base de dados CompLeaks montada, o próximo passo é verificar quais as métricas são mais relevantes para o problema pesquisado, pois no aprendizado de máquina nem sempre uma maior quantidade de métricas é melhor. Por esse motivo optou-se por fazer a seleção das métricas mais relevantes para a utilização nos algoritmos de classificação para a predição de componentes com vazamentos de recursos.

Para fazer essa seleção, será aplicado o algoritmo *Extra Trees*. Esse algoritmo costuma ser utilizado nos estudos que utilizam aprendizado de máquina, como por exemplo em [Calli et al. 2018] e [Ezzini et al. 2018]. As métricas selecionadas irão ser utilizadas para treinar os classificadores apresentados na próxima subseção.

[1] `http://www.virtualmachinery.com/jhawkprod.htm`

[2] A classe é um conceito que encapsula abstrações de dados e componentes que descrevem o conteúdo e o comportamento de entidades do mundo real, representadas por objetos [Pressman and Maxim 2016].

3.2.2. Execução do Treinamento

A base de dados CompLeaks tem rotulado os componentes como contendo vazamentos de recursos ou não e deseja-se prever o rótulo correto para o componente alvo, isso é uma tarefa de classificação que é uma forma de aprendizado supervisionado. Existem muitos modelos de classificação de aprendizado supervisionado e cada um pode executar melhor que o outro dependendo do problema e a base de dados. Nos experimentos, serão usados classificadores que geralmente são utilizados em estudos para predição de erro. Foram incluídos: *support vector machine (SVM)*, *naive bayes*, *deep neural network (DNN)*, *logistic regression*, *random forest* e *k-nearest neighbor (KNN)* [Efendioglu et al. 2018, Li et al. 2019, Manjula and Florence 2019].

Os classificadores listados acima serão treinados utilizando a base de dados CompLeaks. Após isso, será realizada uma análise de qual desses classificadores é o mais eficaz para o problema pesquisado, o mesmo será utilizado no processo de classificação dos componentes para as aplicações recebidas como artefato de entrada.

3.2.3. Código Fonte da App

Para a utilização da abordagem proposta, é recebido como artefato de entrada o código fonte de uma aplicação Android, o qual deverá ter sido implementado na linguagem Java, pois a ferramenta JHawk (utiliza análise estática), que será utilizada para a extração das métricas, é compatível apenas com essa linguagem de programação. Com a intenção de classificar os potenciais componentes com vazamentos de recursos, dessa aplicação, será necessário extrair algumas métricas da aplicação conforme mostrado na próxima subseção.

3.2.4. Extração das Métricas

As métricas apresentadas na Tabela 1 serão extraídas para cada um dos componentes, da aplicação móvel recebida como artefato de entrada. Após isso serão removidos os componentes em que suas classes não referenciam alguma das 33 classes de recursos listadas na Tabela 2, isso tem como objetivo diminuir os falsos positivos, pois se a classe não referencia nenhuma das classes de recursos mapeadas ela não terá vazamentos dos recursos mapeados. A lista de classes de recursos apresentada na Tabela 2 foi extraída a partir da base de dados DroidLeaks. Será utilizado como artefato de entrada para a classificação a lista de componentes da aplicação com seus respectivos valores de métricas, conforme apresentado na próxima subseção.

3.2.5. Classificação dos Componentes

Neste processo será utilizado o classificador selecionado no processo de execução do treinamento. O classificador receberá como entrada a lista de componente gerada na etapa anterior. Após a execução do classificador, a lista de componentes com vazamentos será apresentada no seguinte formato: pacote, classe, componente, parâmetros e retorno. O

Tabela 2. Lista de classes de recurso

ID	Classe de Recurso	ID	Classe de Recurso	ID	Classe de Recurso
CL1	android.database.Cursor	CL12	java.util.Formatter	CL23	java.net.Socket
CL2	android.database.sqlite.SQLiteDatabase	CL13	android.os.PowerManager.WakeLock	CL24	android.os.ParcelFileDescriptor
CL3	java.io.BufferedOutputStream	CL14	android.os.Parcel	CL25	java.io.PipedOutputStream
CL4	java.io.BufferedReader	CL15	java.util.Scanner	CL26	java.util.logging.FileHandler
CL5	java.io.DataOutputStream	CL16	java.io.FilterInputStream	CL27	android.location.LocationListener
CL6	java.io.FileInputStream	CL17	java.io.FilterOutputStream	CL28	android.hardware.Camera
CL7	java.io.FileOutputStream	CL18	java.io.InputStreamReader	CL29	java.io.ByteArrayOutputStream
CL8	java.io.OutputStream	CL19	java.io.ObjectInputStream	CL30	java.io.BufferedWriter
CL9	java.io.InputStream	CL20	java.io.ObjectOutputStream	CL31	java.io.OutputStreamWriter
CL10	org.apache.http.impl.client.DefaultHttpClient	CL21	android.media.MediaPlayer	CL32	android.view.MotionEvent
CL11	java.util.concurrent.Semaphore	CL22	android.net.wifi.WifiManager.WifiLock	CL33	android.net.http.AndroidHttpClient

desenvolvedor utilizará essa lista de componentes para guiar a remoção dos vazamentos de recursos.

4. Trabalhos relacionados

Diferentes abordagens foram propostas para a solução do problema de identificação de vazamento de recursos: [Guo et al. 2013] propõem um método automático para detectar vazamentos de recursos com base em um gráfico de chamada de componente modificado, que lida com os recursos da programação móvel orientada a eventos, analisando os retornos de chamada definidos na estrutura do Android.

[Qian and Zhou 2016] desenvolveram uma nova abordagem para priorizar os casos de teste de acordo com sua probabilidade de causar vazamentos de memória em um determinado conjunto de testes. Em primeiro lugar, é construído um modelo de previsão para determinar se cada teste pode potencialmente levar a vazamentos de memória com base no aprendizado de máquina em recursos de código selecionados. Em seguida, cada caso de teste de entrada é executado parcialmente para obter seus recursos de código e prever sua probabilidade de causar vazamentos. A abordagem sugere que os casos de teste mais suspeitos sejam executados primeiro, a fim de revelar falhas de vazamento de memória o mais rápido possível.

[Jiang et al. 2017] apresentam uma técnica de análise estática chamada SAAD, que detecta vazamento de recursos em aplicações Android por análise sensível ao contexto, a qual combina análise de chamada de componente, análise interprocedimento e análise intraprocedimento, sendo considerado o contexto de chamada ao analisar o destino de uma chamada de componente. Para melhorar a eficiência, os autores se concentraram na análise de caminhos eficazes que estão envolvidos nas operações de utilização/liberação de recursos.

[Hoshieah et al. 2019] desenvolveram uma ferramenta chamada SAALC que é capaz de analisar as atividades (em inglês, activities) do Android e extrair informações valiosas sobre o uso de componentes de retorno de chamada do ciclo de vida. Os resultados mostraram quais componentes de retorno de chamada são implementados e a natureza do código que eles contêm. De igual modo, mostraram a implementação incorreta dos componentes de retorno de chamada e aquisição e liberação incorreta de recursos do sistema em muitos apps Android.

[Toffalini et al. 2019] apresentam um novo método de análise estática que encontra vazamentos de contexto em aplicações nativas Android escritas em Java. Primeiro, os APKs são convertidos em bytecode Java. Segundo, o bytecode Java é analisado com a

Tabela 3. Comparação com os trabalhos relacionados

Artigo	Objetivo	Análise	Detalhes
[Guo et al. 2013]	Detectar vazamentos de recursos	Estática	Programação móvel orientada a eventos e chamada de componente modificado
[Qian and Zhou 2016]	Priorizar casos de teste para a revelação de vazamentos de recursos	Estática e Dinâmica	Aprendizado de máquina
[Jiang et al. 2017]	Detectar vazamentos de recursos	Estática	Análise sensível ao contexto (análise de chamada de componente, análise interprocedimento e análise intraprocedimento)
[Hoshieah et al. 2019]	Detectar vazamentos de recursos	Estática	Componentes de retorno de chamada do ciclo de vida
[Toffalini et al. 2019]	Detectar vazamentos de recursos	Estática	Identificar locais onde contextos podem se tornar acessíveis a partir de campos estáticos ou threads
[Bhatt and Furia 2020]	Detectar e corrigir automaticamente vazamentos de recursos	Estática	Gráfico de fluxo de recursos
Este trabalho	Detectar vazamentos de recursos	Estática	Aprendizado de máquina

ferramenta Julia para identificar locais onde contextos podem se tornar acessíveis a partir de campos estáticos ou threads. Por último, os potenciais vazamentos são analisados sistematicamente para definir a sua gravidade. Por exemplo, se a execução de um componente vazar um contexto, o problema será mais perigoso se o componente contiver loops ou executar bloqueio de E/S, estendendo assim a duração do vazamento.

[Bhatt and Furia 2020] propõem PlumbDroid: uma técnica para detectar e corrigir automaticamente vazamentos de recursos em aplicações Android. PlumbDroid usa análise estática para encontrar rastros de execução que podem vazar um recurso. As informações criadas para detecção também sustentam a construção automática de uma correção - consistindo em operações de liberação realizadas em locais apropriados - que remove o vazamento e não afeta o uso do recurso pela aplicação.

Na Tabela 3 é mostrado um resumo dos trabalhos relacionados. Um dos trabalhos relacionados utiliza análise estática e dinâmica e aprendizado de máquina para priorizar casos de teste com maior probabilidade de revelar vazamentos de recursos. Este trabalho, por sua vez, pretende utilizar métricas extraídas utilizando análise estática e aprendizado de máquina com o objetivo de detectar vazamentos de recursos, considerando uma maior variedade de classes de recursos do que os trabalhos relacionados.

5. Resultados Esperados

Espera-se disponibilizar uma abordagem para a identificação de componentes propensos a ter vazamentos de recursos. Pretende-se disponibilizar uma extensão da base de dados DroidLeaks com várias métricas extraídas no nível de componente, a qual será utilizada para a realização de uma análise sobre o desempenho de técnicas de AM para prever vazamento de recursos em componentes em aplicações Android. Há a expectativa ainda de se disponibilizar uma ferramenta que permita identificar componentes com vazamentos de recursos em aplicações móveis.

Referências

Bhatt, B. N. and Furia, C. A. (2020). Automated repair of resource leaks in android applications. *arXiv preprint arXiv:2003.03201*.

Calli, B., Srinivasan, K., Morgan, A., and Dollar, A. M. (2018). Learning modes of within-hand manipulation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3145–3151. IEEE.

Efendioglu, M., Sen, A., and Koroglu, Y. (2018). Bug prediction of systemc models using machine learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(3):419–429.

El Naqa, I. and Murphy, M. J. (2015). What is machine learning? In *Machine Learning in Radiation Oncology*, pages 3–11. Springer.

Ezzini, S., Berrada, I., and Ghogho, M. (2018). Who is behind the wheel? driver identification and fingerprinting. *Journal of Big Data*, 5(1):9.

Gao, J., Bai, X., Tsai, W.-T., and Uehara, T. (2014). Mobile application testing: a tutorial. *Computer*, 47(2):46–55.

Guo, C., Zhang, J., Yan, J., Zhang, Z., and Zhang, Y. (2013). Characterizing and detecting resource leaks in android applications. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 389–398. IEEE.

Hoshieah, N., Zein, S., Salleh, N., and Grundy, J. (2019). A static analysis of android source code for lifecycle development usage patterns. *Journal of Computer Science*, 15(1):92–107.

Jiang, H., Yang, H., Qin, S., Su, Z., Zhang, J., and Yan, J. (2017). Detecting energy bugs in android apps using static analysis. In *International Conference on Formal Engineering Methods*, pages 192–208. Springer.

Li, Z., Jing, X.-Y., Zhu, X., Zhang, H., Xu, B., and Ying, S. (2019). Heterogeneous defect prediction with two-stage ensemble learning. *Automated Software Engineering*, 26(3):599–651.

Liu, Y., Wang, J., Wei, L., Xu, C., Cheung, S.-C., Wu, T., Yan, J., and Zhang, J. (2019). Droidleaks: a comprehensive database of resource leaks in android apps. *Empirical Software Engineering*, 24(6):3435–3483.

Manjula, C. and Florence, L. (2019). Deep neural network based hybrid approach for software defect prediction using software metrics. *Cluster Computing*, 22(4):9847–9863.

Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2018). *Foundations of machine learning*. MIT press.

Nimbalkar, R. R. (2013). Mobile application testing and challenges. *International Journal of Science and Research*, 2(7):56–58.

Pressman, R. and Maxim, B. (2016). *Engenharia de Software-8ª Edição*. McGraw Hill Brasil.

Qian, J. and Zhou, D. (2016). Prioritizing test cases for memory leaks in android applications. *Journal of Computer Science and Technology*, 31(5):869–882.

Statista (2020a). Installed base of smartphones by operating system from 2015 to 2017. `https://bit.ly/2xEx73E` (Accessed 28 August 2020).

Statista (2020b). Number of available applications in the google play store from december 2009 to june 2020. `https://bit.ly/2mOe6UQ` (Accessed 28 August 2020).

Toffalini, F., Sun, J., and Ochoa, M. (2019). Practical static analysis of context leaks in android applications. *Software: Practice and Experience*, 49(2):233–251.