CBSoft 2024 | SAST | Curitiba – PR | 30/09/2024

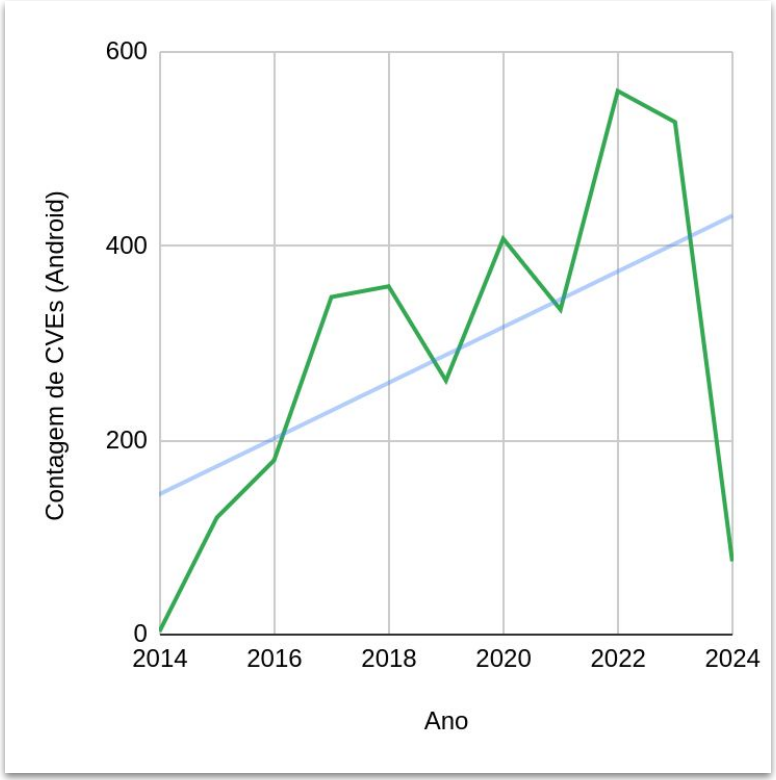
Mutation Testing to Support the Security Testing of Android Applications

Eduardo S. M. de Vasconcelos <vasconcelos.esm@gmail.com>
Marcio E. Delamaro <delamaro@icmc.usp.br>
Simone R. S. Souza <srocio@icmc.usp.br>

Instituto de Ciências Matemáticas e de Computação (ICMC)
Universidade de São Paulo (USP)
São Carlos – SP, Brasil

Motivação

- Dezembro de 2023: mais de 2,4 milhões de apps no Google Play [52]:
  - Muitas dessas aplicações têm casos de uso sensíveis (e.g. saúde, finanças, etc.);
  - Muitas são publicadas por desenvolvedores amadores [3];
- Tendência de aumento do número de CVEs no Android [40];
- Aumento da relevância do Teste de Segurança em Android;
- **Como medir a qualidade desses procedimentos de teste?**



Proposta

Aplicar Teste de Mutação

- **Estudo preliminar** sobre o uso de Teste de Mutação no contexto de Teste de Segurança de apps Android:
  - Propomos **novos operadores** de mutação específicos para segurança de apps Android;
  - Desenvolvemos uma **nova ferramenta** para geração de mutantes aplicando os operadores propostos (`seed-vulns` [54]);
  - Avaliamos a **verossimilhança dos operadores propostos** usando uma ferramenta de análise estática (`mobsfscan` [2]).

Conceitos Iniciais

Apps Android

OEM vs. Usuário

Apps podem ser pré-instalados pelo fabricante (Open Equipment Manufacturer, OEM) ou podem ser instalados pelo usuário, a partir de repositórios externos, como o Google Play.

Android Package (APK)

O APK é o formato mais básico de app Android. Contém todos os recursos necessários ao funcionamento do app, o que inclui bytecode DEX (Dalvik Executable), a ser executado pela máquina virtual Android (Android Runtime, ART).

Android Manifest

Descrição do app em alto nível (componentes de aplicação, permissões e recursos de sistema utilizados). É escrito pelo desenvolvedor em XML e faz parte do APK.

Componentes de Aplicação

Partes do app que implementam, efetivamente, as suas funcionalidades. São descritos no manifest e escritos em linguagem de alto nível (e.g. Java, Kotlin, C/C++) usando a API do Android.

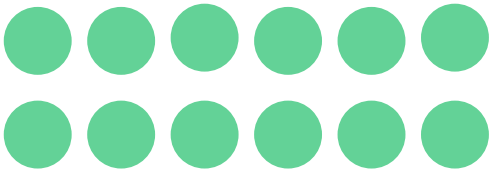
Conceitos Iniciais

Teste de Mutação

Programa

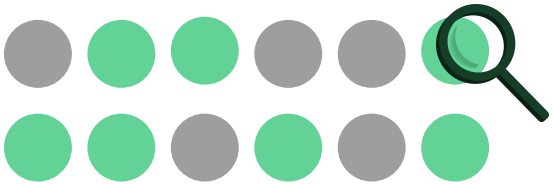
Conjunto de casos de teste de Programa

Pequenas modificações (“mutações”) introduzidas em Programa, para obter versões alternativas (“mutantes”).



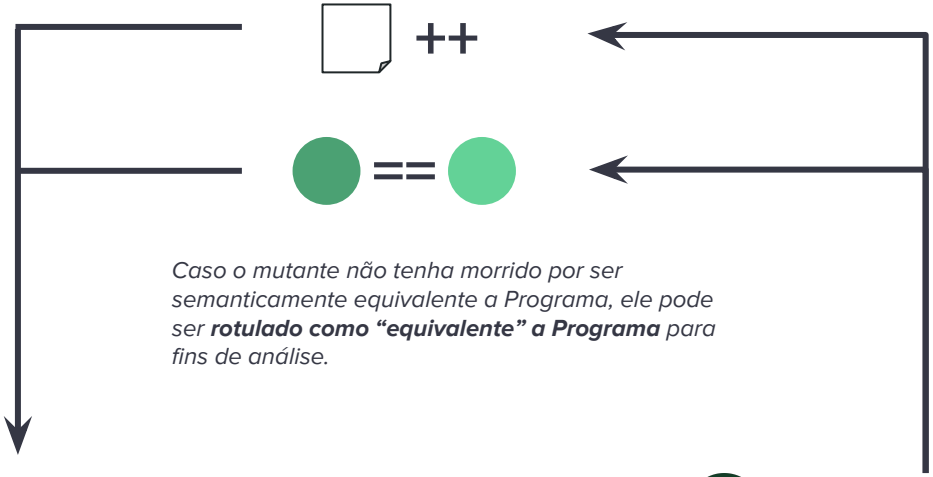
Mutantes de Programa

*Mutantes são **executados** contra o conjunto de casos de teste de Programa.*



*Mutantes cujo comportamento é diferente do de Programa ao ser executado contra o conjunto de casos de teste “morrem”. **Outros não morrem.***

*Cada mutante que não morre é analisado para descobrir **por que o conjunto de casos de teste não foi capaz de “matá-lo”** i.e. de revelar a mutação introduzida.*



*Caso o mutante não tenha morrido por ser semanticamente equivalente a Programa, ele pode ser **rotulado como “equivalente” a Programa** para fins de análise.*

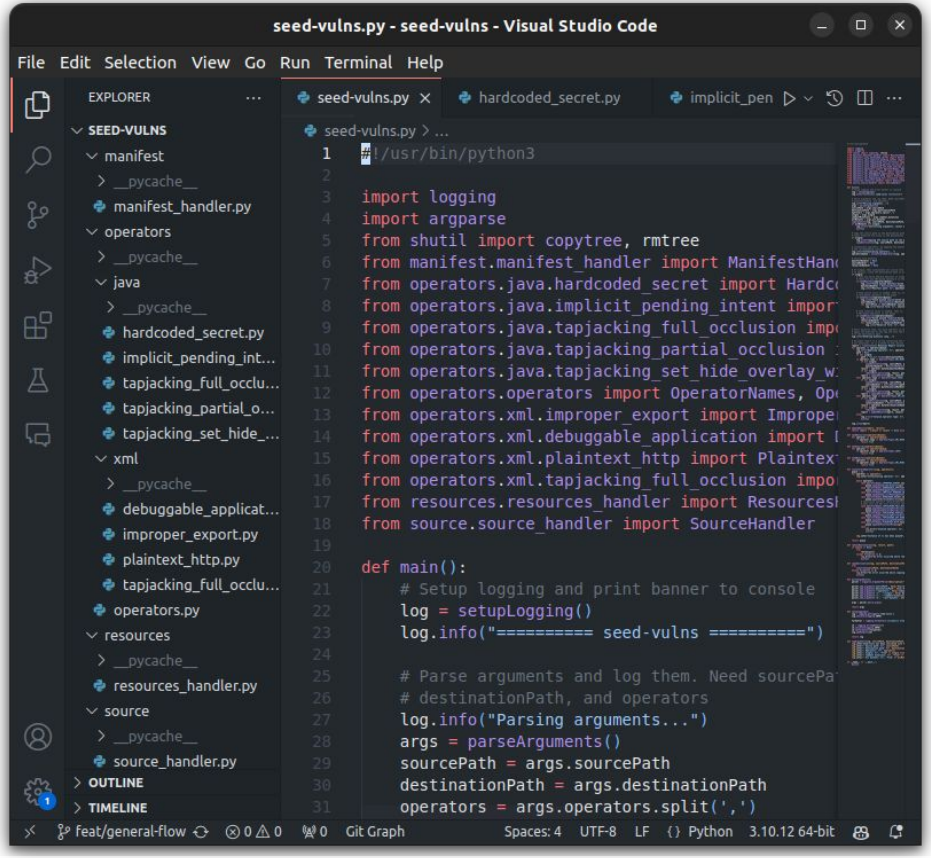
*Caso o mutante não tenha morrido porque o conjunto de casos de teste não foi capaz de revelar a mutação, **melhorias podem ser introduzidas no conjunto de casos de teste.***

Operadores de Mutação Propostos

Operador	Descrição	Java	Kotlin	XML
`ImproperExport`	Exportação desnecessária de componente de aplicação, expondo-o indevidamente. [28]			✓
`DebuggableApplication`	Flag de debug ativada, permitindo *debugging* do app. [28]			✓
`ImplicitPendingIntent`	Mau uso de um mecanismo de IPC (Inter-Process Communication) específico, chamado de *pending intent*. [28]	✓	✓	
`HardcodedSecret`	Segredo (e.g. senha, chave, etc.) *hardcoded* na aplicação. [28]	✓	✓	
`TapJackingFullOcclusion`	Mau uso de proteção contra *tap jacking* (possibilidade 1: `filterTouchesWhenObscured`). [28]	✓	✓	✓
`TapJackingPartialOcclusion`	Mau uso de proteção contra *tap jacking* (possibilidade 2: `FLAG_WINDOW_IS_PARTIALLY_OBSCURED`). [28]	✓	✓	
`TapjackingSetHideOverlayWindows`	Mau uso de proteção contra *tap jacking* (possibilidade 3: `setHideOverlayWindows`). [28]	✓	✓	
`PlaintextHTTP`	Uso de HTTP sem criptografia na camada de transporte (TLS). [28]			✓

Ferramenta: `seed-vulns`

- Disponível publicamente no GitHub [54];
- Desenvolvida em **Python;**
- Implementa todos os operadores de mutação aqui propostos;
- Recebe como entradas um **app Android** (repositório de código-fonte) e uma l**ista de operadores de mutação** a aplicar;
- Gera mutantes para apps Android escritos em **Java e/ou Kotlin de forma transparente;**
- Gera, ainda, um **relatório de mutação,** especificando as mutações que foram aplicadas ao app de entrada;
- **Limitação:** por ora, apenas gera mutantes. Versões futuras suportarão análise de mutantes propriamente dita.



Estudo Experimental (I)

Questão de Pesquisa

RQ: *Do the proposed mutation operators represent real-world vulnerabilities found in Android apps?*

Se as mutações aplicadas forem detectadas como vulnerabilidades ao executar testes de segurança contra os mutantes, isso atesta a verossimilhança dos operadores de mutação.

Hipóteses

H_0: *The mutations are not detected as vulnerabilities.*

H_A: *The mutations are detected as vulnerabilities.*

Como os operadores de mutação são independentes, avaliamos as hipóteses com relação a cada operador de maneira independente.

Sujeitos

10 apps Android *open source,* desenvolvidos de maneira intencionalmente vulnerável, para fins de educação.

Métrica

Para cada app sujeito do experimento:

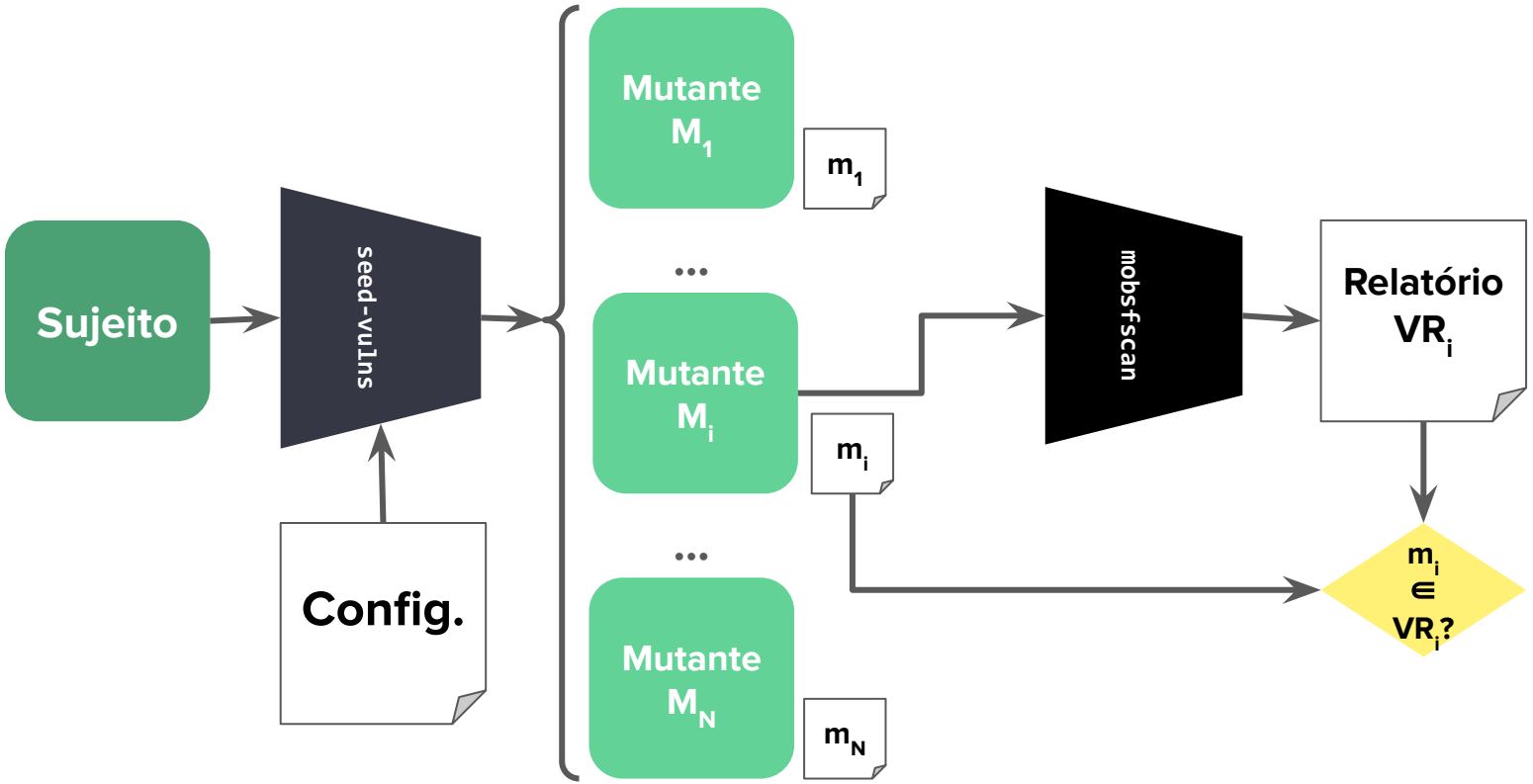
$$\frac{\#\text{Mutações detectadas como vulnerabilidades}}{\#\text{Mutações introduzidas}}$$

e.g. 10 mutantes (i.e. 10 mutações) gerados a partir do app A; 8 das quais detectadas como vulnerabilidades ⇒ 8/10 = 0,8

Ferramenta de Teste

`mobsfscan` [2]

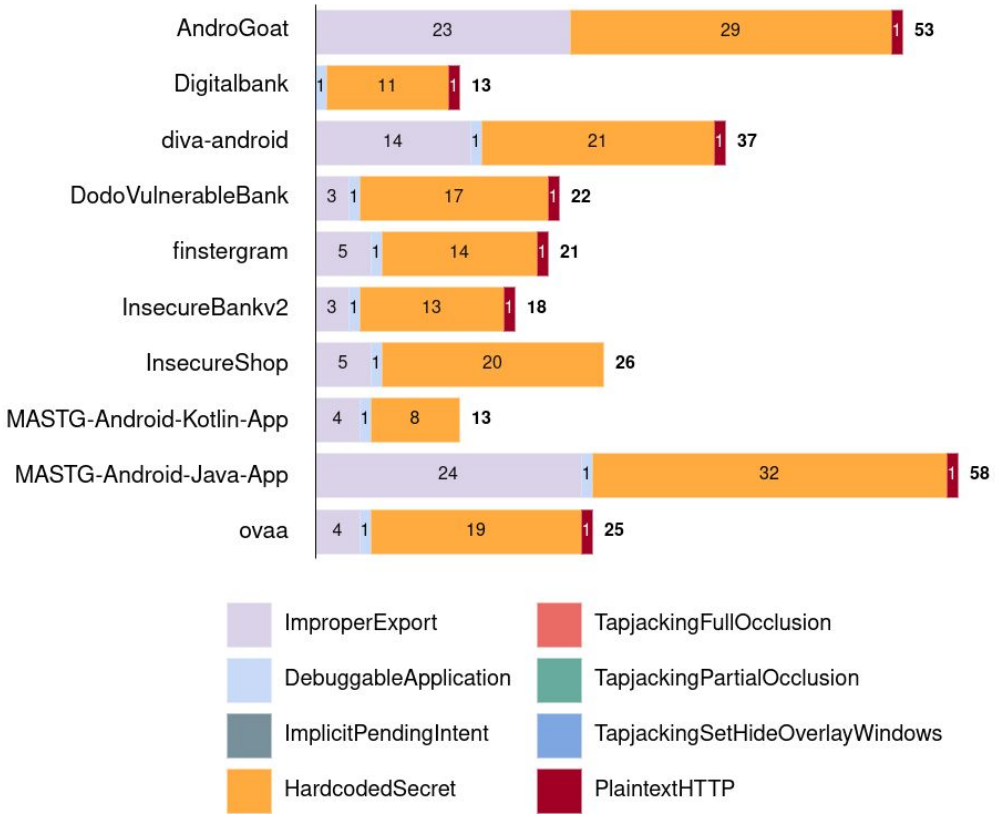
Estudo Experimental (II)



Resultados e Discussão (I)

Características dos Mutantes

- Total de mutantes gerados: **286;**
- **LIMITAÇÃO:** operadores **não representados:** `ImplicitPendingIntent`, `Tapjacking*`;
- Operadores **super representados:** `ImproperExport`, `HardcodedSecret`.



Resultados e Discussão (II)

Sujeito	#Mutantes	ImproperExport ?	HardcodedSecret ?	PlaintextHTTP	DebuggableApplication
AndroGoat	53	0/23 (0.0)	0/29 (0.0)	1/1 (1.0)	0/0 (N/A)
Digitalbank	13	0/0 (N/A)	11/11 (1.0)	1/1 (1.0)	1/1 (1.0)
diva-android	37	0/14 (0.0)	21/21 (1.0)	1/1 (1.0)	1/1 (1.0)
DodoVulnerableBank	22	0/3 (0.0)	17/17 (1.0)	1/1 (1.0)	1/1 (1.0)
finstergram	21	0/5 (0.0)	0/14 (0.0)	1/1 (1.0)	1/1 (1.0)
InsecureBankv2	18	0/3 (0.0)	13/13 (1.0)	1/1 (1.0)	1/1 (1.0)
InsecureShop	26	0/5 (0.0)	0/20 (0.0)	0/0 (N/A)	1/1 (1.0)
MASTG-Android-Kotlin-App	13	0/4 (0.0)	0/8 (0.0)	0/0 (N/A)	1/1 (1.0)
MASTG-Android-Java-App	58	0/24 (0.0)	32/32 (1.0)	1/1 (1.0)	1/1 (1.0)
ovaa	25	0/4 (0.0)	19/19 (1.0)	1/1 (1.0)	1/1 (1.0)
	Média	**0.0**	**0.6**	**1.0**	**1.0**
	Desvio Padrão	**0.0**	**4.899**	**0.0**	**0.0**

Detectabilidade das Mutações

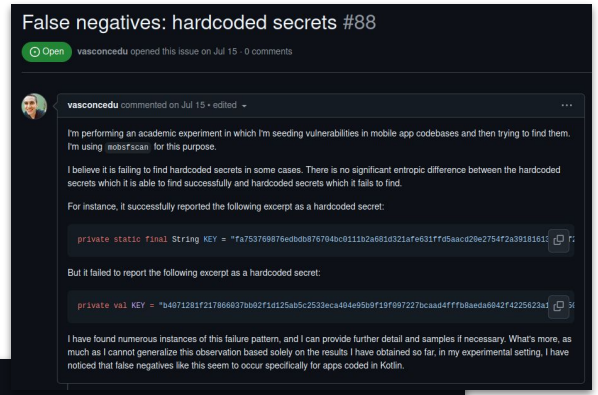
Resultados e Discussão (III)

`ImproperExport?`

- Operador inadequado?
- As mutações **foram inseridas corretamente;**
- Mesmo instâncias pré-existentes não foram detectadas;
- ***Improper exports* têm uma natureza inerentemente semântica, dependem de contexto e não podem ser facilmente detectados por análise estática simples;**
- **LIMITAÇÃO:** impossibilidade de averiguar a verossimilhança do operador `ImproperExport`.

`HardcodedSecret?`

- As mutações **foram inseridas corretamente;**
- `mobsfscan` realmente apresentou falsos negativos: ***regex* de *hardcoded secrets* em Kotlin mal construída!**




```
331   - id: android_kotlin_hardcoded
332     message: >-
333       Files may contain hardcoded sensitive information like usernames,
334       passwords, keys etc.
335     input_case: lower
336     pattern: >-
337       (password\s*=\s*[\'|\"].{1,100}[\'|\"]\s{0,5})|(pass\s*=\s*[\'|\"]
338     severity: WARNING
339     type: Regex
```

Ameaças à Validade & Trabalhos Futuros

Ameaças à Validade

- Limitação: impossibilidade de averiguar a verossimilhança dos operadores `ImplicitPendingIntent`, `Tapjacking*` e `ImproperExport`;
- Sujeitos contêm vulnerabilidades fabricadas e podem não corresponder a aplicações Android reais (i.e. sem cunho educacional);
- Um único autor revisou os resultados e esse processo foi semi-automatizado.

Trabalhos Futuros

- Avaliar os operadores propostos de maneira mais abrangente;
- Avaliar outras ferramentas de análise estática de segurança em busca de *design flaws;*
- Propor operadores de mutação de segurança adicionais para Android;
- Melhorar `seed-vulns`.

Obrigado!

Perguntas?

Eduardo S. M. de Vasconcelos <vasconcelos.esm@gmail.com>
Marcio E. Delamaro <delamaro@icmc.usp.br>
Simone R. S. Souza <srocio@icmc.usp.br>

Instituto de Ciências Matemáticas e de Computação (ICMC)
Universidade de São Paulo (USP)
São Carlos – SP, Brasil