UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Detecção vs Evasão: Uma avaliação do cenário de detecção do root no Android

Trabalho de Graduação

Aluno: Vinicius Moraes
Orientadora: Jéssyka Flavyanne Ferreira Vilela
Área: Segurança da Informação

RECIFE
2020

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA

Vinicius Moraes

Detecção vs Evasão: Uma avaliação do cenário de detecção do root no Android

Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientadora: Jéssyka Flavyanne Ferreira Vilela

RECIFE
2020

AGRADECIMENTOS

Agradeço a Deus pela minha vida.

À minha família por me apoiarem direta e indiretamente nos caminhos que escolhi seguir, em especial na minha vinda para Recife.

À professora Jéssyka, por suas orientações, revisões e dicas que proporcionaram a criação deste trabalho.

Ao professor Leopoldo, por ter aceitado fazer parte da banca avaliadora deste estudo.

Aos professores do CIn que me mostraram as belezas que existem na computação, com destaque para Sérgio Soares, Juliano Iyoda, Márcio Cornélio e Luciano Barbosa por suas aulas incríveis.

A todas as pessoas que fazem do CIn um ambiente aconchegante e sensacional.

Aos diversos amigos que fiz durante a graduação, em especial ao pessoal do Grupo Alpha: Moabe, Thomas, Jailson, Matheus, Geimison e Felipe, que me acompanharam durante a jornada na faculdade.

Aos meus amigos porquinanos/prepianos: Matheus, Nunes, Bruno, João, Show, Ramom, Rogério. Ainda que tenhamos se afastado, sou muito grato a vocês pelos bons tempos que tivemos!

A SuperChoque, Gabi e Lino por terem ajudado diretamente neste estudo.

À Rkos por ter clareado um pouco mais a minha visão sobre a ciência, além de ter me divertido.

Sou grato também às diversas pessoas que são ou já foram meus amigos a um tempo mais distante, em especial: Auguuusto, Biel, Cauê, Gust, Sté, EMDB, Dioogo, Drê, (é o) Alexandre, Tom, Guilherme, Viniiiiicius, Veras, o Júnior, Nicolas, Pedro Leal, Wesley, Matheus, Vitor, John Uchiha, Dialisson, Carlos, Jeison.

Por fim, a você que é uma pessoa espetacular e eu não citei, provavelmente por questões de tempo ou espaço, obrigado!

“Whether you think you can, or think you can't... you're right.”

Henry Ford

RESUMO

O Android é hoje o sistema operacional mais utilizado no mundo. Esse grande número de usuários faz dele um sistema muito visado por criminosos digitais, gerando desafios para os desenvolvedores de aplicativos móveis manterem a segurança de seus usuários. Um desses desafios surge quando as aplicações são executadas em aparelhos que possuem acesso *root*. Root é o nome do superusuário do Android que se mal utilizado pode acarretar no comprometimento do aparelho por parte de atacantes. Além disso, existem ferramentas para auxiliar na exploração de aplicativos, que requerem dispositivos com *root*. Assim, realizar a detecção do acesso *root* se tornou uma medida de segurança adotada por alguns aplicativos. Adicionalmente, nos últimos anos surgiu uma nova forma de obter e esconder o acesso *root*, através do *software* Magisk, que se tornou muito popular e consegue burlar mecanismos de detecção. Dessa forma, o foco deste trabalho é investigar as técnicas de evasão e detecção de *root* utilizadas na prática, avaliar a eficácia da detecção contra os métodos de evasão, em especial o Magisk, e propor possíveis melhorias para a detecção. Para atingir esse objetivo as seguintes medidas foram realizadas: uma investigação da literatura sobre o tema, um *survey* com usuários e desenvolvedores do Android, uma análise de eficácia dos aplicativos que tem o objetivo de detectar o *root*, e também dos aplicativos mais baixados no Google Play e, finalmente, a apresentação de aprimoramentos para algumas das técnicas de detecção. Espera-se que este projeto ajude na compreensão da disputa entre detecção e evasão, e permita que aplicações mais seguras possam ser desenvolvidas.

Palavras-chave: Android, Root, Detecção, Evasão, Magisk, Eficácia, Análise.

ABSTRACT

Android is now the most widely used operating system in the world. This large number of users makes it a very targeted system for digital criminals, creating challenges for mobile application developers to maintain the security of their users. One of these challenges arises when applications are run on devices that have root access. Root is the name of the Android superuser that, if misused, can result in compromising the device by attackers. In addition, there are tools to assist in exploiting applications, which require rooted devices. Therefore, performing root access detection has become a security measure adopted by some applications. Additionally, in the last few years, a new way of obtaining and hiding root access has emerged, through the Magisk software, which has become very popular and can bypass detection mechanisms. So, the focus of this work is to investigate the evasion and root detection techniques used in practice, to evaluate the effectiveness of detection against evasion methods, especially Magisk, and to propose possible improvements for detection. To achieve this objective, the following measures were taken: an investigation of the literature on the subject, a survey of Android users and developers, an analysis of the effectiveness of applications that aim to detect root, and also of the most downloaded applications on Google Play and, finally, the presentation of improvements for some of the detection techniques. It is hoped that this project will help to understand the dispute between detection and evasion, and allow safer applications to be developed.

Keywords: Android, Root, Detection, Evasion, Magisk, Effectiveness, Analysis.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1. Introdução

Este capítulo visa fornecer uma visão sobre o problema da pesquisa bem como os objetivos que serão abordados ao longo do trabalho.

1.1 Contexto

Estima-se que hoje existam mais de 3.5 bilhões de pessoas usando smartphones ao redor do mundo [1], desse total, aproximadamente 74% utiliza o Android como seu sistema operacional [2]. Entretanto, esse alto número de usuários torna o sistema visado não só para desenvolver novos aplicativos e jogos, como também para a criação de novos softwares maliciosos, do inglês *malware* (*mal*icious soft*ware*)[3].

Esses *malwares* podem ter os mais diversos objetivos, como por exemplo: roubar informações do usuário, fornecer acesso remoto a um *hacker*, infectar outros usuários da mesma rede, minerar criptomoedas, exibir anúncios que geram lucro ao atacante [9].

Em paralelo a esse crescimento do sistema, uma prática que se tornou comum entre alguns usuários do Android é a de realizar o procedimento de *rooting*, ou seja, ativar o acesso de superusuário (*root)* em seus aparelhos. Foi estimado que em 2017 o número de dispositivos *rooteados* (ou seja, com acesso *root*) na Indonésia chegava a ser de 12%, esse número aumenta quando olha-se para a Moldávia com 15% ou ainda para a Venezuela com 26% [4].

Alguns dos motivos que levam os usuários a desejarem o *root* são: utilizar um determinado aplicativo, personalizar mais o Android (através de novos temas e funcionalidades) ou ainda desinstalar *bloatwares*, que são aplicativos indesejados que podem vir instalados no sistema e, às vezes, não são completamente removíveis por métodos tradicionais [13]. Contudo, muitos desses usuários acabam não percebendo as complicações de segurança que essa prática pode causar.

Os métodos para obter o acesso *root* em um Android podem ser divididos, de acordo com a abordagem que é utilizada, em dois grupos: *post-boot* e *pre-boot* [20]. Os métodos do tipo *post-boot* são aqueles que exploram características e vulnerabilidades no próprio Android, depois que ele já foi iniciado. Esses métodos são muito específicos e com isso menos populares, e portanto, não são o foco deste trabalho.

Por outro lado, os métodos do tipo *pre-boot* [20] são aqueles que envolvem procedimentos antes da inicialização do Android. Esse segundo grupo fornece soluções mais genéricas (funcionando para a maioria dos dispositivos), e portanto, são mais populares. Nesse grupo, existia um *software* de código fechado

extremamente popular e que por muito tempo foi o mais utilizado, o SuperSU,[1] que também foi alvo de todos os trabalhos relacionados encontrados que serão abordados na Seção 1.4. No entanto, o SuperSU acabou perdendo espaço para uma nova ferramenta de código aberto e, posteriormente, foi descontinuado [28]. Essa nova ferramenta é o Magisk que atualmente é a forma mais popular de adquirir o acesso *root* [17]. A Figura 1 apresenta uma comparação da popularidade entre as duas ferramentas nas pesquisas do Google.

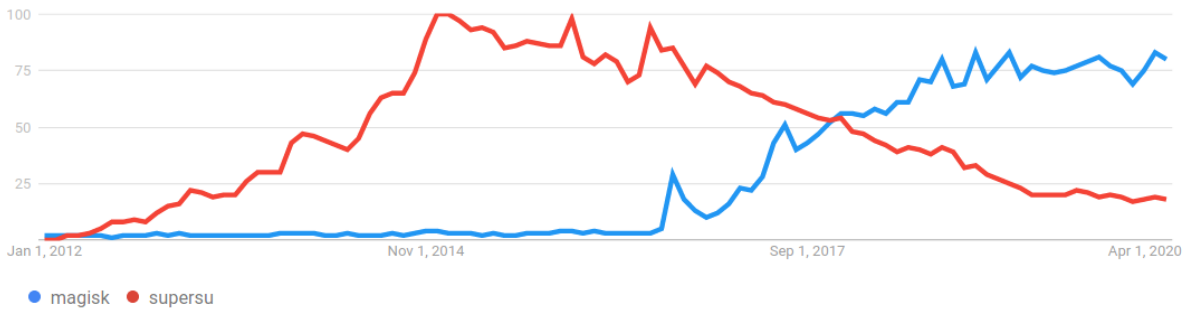


Figura 1: Comparação de popularidade entre o Magisk e o SuperSU.
Fonte: Google Trends.[2]

1.2 Definição do Problema

Quando um aplicativo pode ser executado normalmente em dispositivos *rooteados* surgem dois principais problemas. O primeiro é em relação a segurança do próprio usuário. Como o acesso *root* permite que um programa obtenha privilégios muito elevados no sistema, existe o risco de que vazamentos de dados aconteçam, pois se torna possível ler arquivos privados de outros aplicativos, além da possibilidade de acesso aos dados da memória e de rede deles. Dessa forma, um *malware* pode obter *tokens* de sessão de outros aplicativos e executar operações indesejadas neles, como roubar mensagens privadas de redes sociais ou ainda realizar operações financeiras sem a autorização do usuário [5].

O segundo problema se refere a segurança do próprio serviço entregue pelo aplicativo, já que existem ferramentas para fraude que necessitam de privilégios de *root*, como por exemplo a possibilidade de burlar compras internas dentro dos aplicativos.[3]

[1] Disponível em < https://download.chainfire.eu/1220/ >. Acessado em 13 de junho de 2020.
[2] Disponível em < https://trends.google.com.br/trends/explore?date=2012-01-01%202020-05-01&q=supersu,magisk >. Acessado em 13 de junho de 2020.
[3] Disponível em < https://medium.com/@freedom.apk/how-to-get-paid-features-of-apps-for-free-90ac1000dfc9 >, acessado em 08 de setembro de 2020.

Por conta dessas ameaças que os aplicativos sofrem ao serem executados em aparelhos *rooteados*, alguns deles começaram a implementar técnicas de detecção de *root*.

No contexto deste trabalho, detecção é:

o ato de se utilizar de técnicas para procurar sinais que indicam a presença do root em um dispositivo Android.

Exemplos de técnicas para detecção são apresentadas no Capítulo 2. Em alguns casos, quando o *root* é detectado, apenas é exibida uma mensagem ao usuário, alertando-o sobre o risco que a aplicação está sendo exposta. Em outros, a execução simplesmente é interrompida, impossibilitando que o usuário a utilize.

Embora a detecção seja importante para proteger os usuários que não tem consciência dos perigos de estarem com o *root* habilitado em seus aparelhos, existem outros que aceitam os riscos e querem que seus dispositivos *rooteados* continuem executando todos aplicativos normalmente [13].

Afinal, o acesso *root* não é completamente ruim. Existem vantagens para pesquisadores, desenvolvedores e até mesmo usuários avançados que desejam ter um maior controle sobre seu próprio aparelho. Com esse alto nível de privilégios, é possível modificar parâmetros do *kernel* do Android, em busca de mais desempenho ou economia de energia, ou ainda realizar monitoramentos avançados de rede [15]. Por isso, surgiram métodos de evasão.

Neste trabalho, evasão se refere:

a ação de esconder o acesso root existente em um aparelho Android.

Métodos de evasão (descritos no Capítulo 2) podem ser de fácil uso para o usuário final, como algumas funcionalidades fornecidas pelo Magisk, e também podem ser mais complexos, como se utilizar de *hooking* (ou seja, interceptar e modificar dados)[4] em funções responsáveis pela detecção.

Esse conflito de interesses, entre aqueles que desejam detectar o acesso *root* e aqueles que desejam escondê-lo, gera uma disputa incessante entre dois lados, que é o contexto deste trabalho (detecção vs evasão).

1.3 Objetivos

Este trabalho tem como objetivo realizar uma investigação do estado da arte e da prática das técnicas para detecção de *root*, avaliar a eficácia delas contra os métodos de evasão e aperfeiçoar a detecção. Para alcançar esse objetivo, os seguintes objetivos específicos foram definidos:

1. Revisar a literatura em busca de técnicas de evasão e detecção de *root* existentes;

[4] Disponível em < https://en.wikipedia.org/wiki/Hooking >. Acessado em 2 de abril de 2020.

2. Conduzir *surveys* com usuários e desenvolvedores Android para identificar técnicas de detecção e evasão utilizadas na prática;
3. Analisar aplicativos para avaliar a eficácia das técnicas de detecção implementadas atualmente contra as formas de evasão identificadas;
4. Encontrar comportamentos ou características de aparelhos *rooteados* que podem ser utilizadas para melhorar a detecção.

1.4 Trabalhos Relacionados

Na literatura existem alguns trabalhos que tratam sobre a detecção de acesso *root* no Android. A Tabela 1 apresenta as principais similaridades e diferenças entre este trabalho e outros que foram investigados.

Critério\Trabalho	**EVANS et al., 2015**	**SUN et al., 2015**	**NGUYEN-VU et al., 2017**	**Este Trabalho (2020)**
Local de publicação	*Proceedings of the 13th ACM International Symposium on Mobility Management and Wireless Access.*	*Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices.*	Hindawi: Security and Communication Networks	Centro de Informática da Universidade Federal de Pernambuco.
Foco da pesquisa	Evasão e detecção de *root*	Procedimento de *rooting*, Evasão e detecção de *root*	Evasão e detecção de *root*	Evasão e detecção de *root*
Métodos de Evasão utilizados	AndroPoser, RootCloak, Hide My Root	RDAnalyzer	RootCloak, Renomear binário su	Magisk
Fonte de informação sobre as técnicas de detecção e evasão encontradas	Pesquisa na literatura, Análise estática	Pesquisa na literatura, Análise estática	Pesquisa na literatura, Análise estática	Pesquisa na literatura, Survey com usuários e desenvolvedores, Análise de eficácia da detecção dos aplicativos

Critérios de seleção para análise de aplicativos	Não divulgado.	Aplicativos listados pelo RootCloak.[5]	Busca pelo termo “root checker” no Google Play.	Buscas por “root detect”, “root check”, “root verity”, “magisk check”, “magisk detect” no Google Play, Aplicativos com maior número de downloads.
Quantidade de aplicativos avaliados	16 aplicativos de antivírus e outros 19 de gerenciamento de dispositivos móveis.	30 aplicativos com detecção listados pelo RootCloak	92 aplicativos de detecção de *root*	136 aplicativos de detecção de *root* e 853 aplicativos mais baixados
Melhoria para a detecção	Não apresenta melhorias	Sugere duas abordagens contra evasão por *hooking*	Cria aplicativo que reúne técnicas encontradas	Sugere duas abordagens contra Magisk

Tabela 1: Comparação entre este trabalho e outros relacionados.
Fonte: O autor.

No trabalho de Evans et al. (2015), os autores realizam uma pesquisa literária que os leva a concluir que na época não existia outro artigo com foco na checagem de *root* no Android. Os autores então escolheram, sem divulgar os critérios de seleção, 16 aplicativos de antivírus e outros 19 de gerenciamento de dispositivos móveis (do inglês *mobile device management*) para realizar uma análise estática. Eles identificaram que muitos aplicativos apenas realizavam uma busca de arquivos. Além disso, poucos utilizavam código nativo (feito com C ou C++ utilizando o Android NDK), que poderia gerar uma maior dificuldade de evasão, para realizar as checagens.

De acordo com os resultados obtidos em Evans et al. (2015), os aplicativos analisados também não implementavam ofuscação ou a faziam de maneira simples e sem criptografar strings. A não utilização da ofuscação facilitou bastante a realização de análises estáticas e consequentemente a evasão das técnicas de detecção implementadas pelos aplicativos.

Ao término das análises os autores classificaram os métodos de detecção encontrados em 3 tipos: checagem da presença de arquivos, checagem da

[5] Disponível em < https://repo.xposed.info/module/com.devadvance.rootcloak >, acessado em 31 de maio de 2020.

configuração geral do dispositivo, checagem de recursos e características do sistema. O trabalho também apresenta uma biblioteca que foi criada pelos próprios autores, chamada de AndroPoser, que era injetada nos aplicativos e servia para realizar o *hooking* de funções implementadas em código nativo.

Finalmente, os métodos de detecção abordados em Evans et al. (2015) foram testados contra o AndroPoser, em conjunto com dois aplicativos que serviam para esconder o *root* e atualmente já não são mais mantidos, o RootCloak[6] e o Hide My Root,[7] sendo assim capazes de burlar todas as detecções de *root* que foram encontradas.

Sun et al. (2015) realizaram uma pesquisa literária a fim de entender de quais formas a detecção de *root* poderia ocorrer. Essa pesquisa teve ênfase nos modos como o procedimento de *rooting* poderia ser realizado e suas consequentes modificações no Android.

Os autores realizaram uma análise estática em 30 aplicativos que faziam detecção de *root* e eram listados pelo RootCloak.[8] A partir da análise estática, os autores identificaram as técnicas utilizadas por esses 30 aplicativos e desenvolveram uma ferramenta nomeada de RDAnalyzer. Essa ferramenta realizava o *hooking* de funções Java e também das nativas utilizando o Cydia Substrate,[9] um framework para Android desenvolvido por Jay Freeman (também conhecido como saurik), para que as evasões funcionassem em ambos os casos.

Em seguida, os autores selecionaram aplicativos populares que necessitavam de *root* (gerenciadores de arquivos, backup) e alguns outros que surgiram de buscas por “root check” e “root verify” no Google Play, excluíram aqueles que os autores não consideraram válidos para a pesquisa (por serem de línguas diferentes de inglês e espanhol, específicos para um modelo de aparelho, pagos) e com isso obtiveram um total de 152 aplicativos que foram instalados em um aparelho *rooteado*.

Ao instalar cada um dos 152 aplicativos, eles foram executados com e sem o RDAnalyzer para verificar se suas detecções estavam sendo burladas corretamente. Se um aplicativo continuasse detectando o *root*, os autores realizavam uma análise estática nele e então atualizavam o RDAnalyzer com a nova técnica encontrada.

Todas as análises realizadas permitiram que os autores agrupassem as técnicas de detecção identificadas por eles em 7 tipos: checagem de aplicativos instalados, checagem de arquivos, checagem da tag BUILD, checagem de propriedades do sistema, checagem de permissão de diretórios, checagem de processos, serviços e tarefas, checagem de execução de comandos.

---

[6] Disponível em < https://github.com/devadvance/rootcloak/ >, acessado em 31 de maio de 2020.
[7] Disponível em < http://web.archive.org/web/20170222032636/https://play.google.com/store/apps/details?id=com.amphoras.hidemyroot >, acessado em 31 de maio de 2020.
[8] Disponível em < https://repo.xposed.info/module/com.devadvance.rootcloak >, acessado em 31 de maio de 2020.
[9] Disponível em < http://www.cydiasubstrate.com/ >, acessado em 31 de maio de 2020.

Após a execução dos aplicativos em conjunto com o RDAnalyzer, que também coletava quais técnicas eram utilizadas por cada um, os autores notaram que quase 67% dos aplicativos não empregavam mais do que 3 dos 7 métodos de detecção identificados. Além disso, ao final do trabalho todos os aplicativos analisados tiveram suas técnicas de detecção completamente burladas pelo RDAnalyzer. Os autores terminam sugerindo duas novas possíveis abordagens para a detecção contra métodos de evasão que se utilizavam de *hooking* e, finalmente, concluíram que a detecção estava em desvantagem na corrida contra a evasão.

O estudo de Nguyen-Vu et al. (2017) tem como objetivo descobrir técnicas de detecção de *root* por meio de uma pesquisa literária nos artigos da época e, posteriormente, uma análise estática e manual de cada um dos 92 primeiros aplicativos que apareciam como resultado da pesquisa pelo termo “root checker” no Google Play.

Utilizando o RootCloak, eles configuraram palavras-chaves encontradas nas análises dos 92 aplicativos e conseguiram burlar a detecção de 89 deles. Os autores também escolheram outros 18 aplicativos, só que esses eram financeiros e faziam a checagem de *root* através de código nativo.

Segundo o estudo de Nguyen-Vu et al. (2017), a única forma viável de descobrir funções de detecção nativas na época era através do Cydia Substrate, que já havia sido descontinuado. Então, os autores afirmam que nesses casos a evasão só poderia ocorrer através do uso de engenharia reversa individualmente em cada aplicação, o que seria um processo lento e custoso. No entanto, segundo eles por tentativa e erro, ainda foi possível burlar a detecção de 10 desses 18 aplicativos financeiros, apenas renomeando o binário su.

Finalmente, as técnicas encontradas pelos autores foram agrupadas em 6 tipos: checagem de aplicativos instalados, checagem de arquivos, checagem de processos, serviços e tarefas, checagem de execução de comandos, checagem de propriedades do sistema, checagem de permissões de diretórios. Aqui houve uma categoria a menos do que o estudo de Sun et al. (2015) pois os autores incluíram a checagem da tag BUILD na checagem de propriedades do sistema. Adicionalmente, também foi desenvolvido um aplicativo que unia todas as técnicas de detecção encontradas, chamado de S4URC Root Checker, e que foi lançado no Google Play.

Uma das principais diferenças no cenário deste trabalho, em relação aos que foram analisados, acontece por causa do surgimento do Magisk e a descontinuação do SuperSU. De forma que ambos os eventos alteraram o cenário de detecção ao ponto de tornar muitas implementações obsoletas, um exemplo é o aplicativo S4URC Root Checker que implementa todas as técnicas do estudo de Nguyen-Vu et al. (2017) e que atualmente não consegue detectar o Magisk.

Então, diferente dos trabalhos relacionados que realizaram análises estáticas para identificar os tipos de checagens de *root*, este trabalho realiza análises nos aplicativos para avaliar o quão eficazes são as atuais detecções implementadas por eles contra o Magisk.

Outra distinção deste trabalho em relação aos demais é a apresentação de melhorias para a detecção, que se mostraram eficazes contra o Magisk, como é possível observar no Capítulo 5.

Finalmente, houve a condução de um *survey,* a fim de entender os mecanismos de evasão e detecção de *root* utilizados na prática. A escolha do *survey* ocorreu por ele fornecer um contato direto com usuários e desenvolvedores Android. Este método de pesquisa é interessante para reunir múltiplas visões sobre o atual cenário, além de fornecer um alcance maior de participantes do que uma entrevista.

1.5 Metodologia

Este estudo utiliza o método de pesquisa exploratória, visto que ele tem como foco avaliar se as atuais implementações das técnicas de detecção ainda podem ser aplicadas como medidas eficazes para identificar aparelhos *rooteados*.

A pesquisa foi desenvolvida a partir de dados coletados por meio de revisões da literatura, um *survey* com usuários e desenvolvedores Android e estudos empíricos com aplicativos em aparelhos *rooteados*.

Para as revisões da literatura, foram investigados trabalhos anteriores nessa temática que foram discutidos na Seção 1.4. Adicionalmente, também foram realizadas consultas em livros e sites, que serviram de fundamento para o Capítulo 2.

O *survey* e os estudos empíricos têm seus procedimentos metodológicos detalhados, respectivamente, nos Capítulos 3 e 4.

1.6 Ameaças à Validade

Esta seção apresenta ameaças à validade dos resultados, que foram identificadas neste trabalho.

1.6.1 Validade de Constructo

Validade de *constructo* verifica se questões que serão interpretadas, por pessoas externas à pesquisa, representarão com clareza os objetivos planejados. A fim de reduzir essa ameaça, o *survey,* explicado no Capítulo 3, foi construído de forma iterativa, obtendo-se assim várias visões antes de ser divulgado. Ademais, as respostas obtidas não demonstraram sinais de que houve um problema relevante de entendimento, por parte dos participantes. Portanto, acredita-se que a validade de *constructo* não afetou os estudos empíricos do Capítulo 4 ou as propostas de melhorias do Capítulo 5, visto que ambos não envolveram a participação de pessoas externas.

1.6.2 Validade Interna

Esse aspecto avalia as chances de um fator que está sendo investigado, ser influenciado por causas desconhecidas. Para minimizar os riscos desse fator no *survey*, muitas perguntas foram configuradas para receberem respostas abertas. Já para reduzir esse problema nos estudos empíricos com os aplicativos e nas propostas de melhorias para a detecção, foram considerados os resultados obtidos na revisão da literatura e no *survey*.

1.6.3 Validade Externa

Esse fator visa verificar a relevância do estudo quando ele é generalizado. Em relação ao *survey*, o número de participantes envolvidos pode ser considerado limitado e pequeno considerando o número de usuários e desenvolvedores Android existentes. Contudo, os resultados obtidos no *survey* estão alinhados com as conclusões extraídas dos estudos empíricos realizados no Capítulo 4, que procurou analisar um grande número de aplicativos em relação a outros estudos (o que é detalhado na Tabela 1). Já para as melhorias da detecção, houveram testes em diversos ambientes diferentes (como é mostrado nas Tabelas 3 e 4).

1.6.4 Validade de Confiança

A validade de confiança avalia a reprodutibilidade dos resultados alcançados na pesquisa. A fim de ajudar na replicação do *survey*, todas as perguntas utilizadas e os principais pontos em relação a sua aplicação foram detalhados no Capítulo 3 e no Apêndice A. Para os estudos empíricos, os aplicativos utilizados, os procedimentos realizados e os ambientes de teste foram detalhados no Capítulo 4 e nos Apêndices B e C. E finalmente, para as melhorias propostas, os processos realizados e o código fonte completo do aplicativo prova de conceito foram apresentados no Capítulo 5.

1.7 Estrutura do Documento

O restante do documento segue a organização a seguir:

- O Capítulo 2 realiza uma revisão de literatura e aborda conceitos importantes para o entendimento deste trabalho;
- O Capítulo 3 detalha o protocolo do *survey* e seus resultados;
- O Capítulo 4 apresenta dois estudos empíricos que avaliam a efetividade da detecção do acesso *root*;
- O Capítulo 5 sugere abordagens para melhorar a detecção;
- O Capítulo 6, por fim, expõe as conclusões e considerações finais do trabalho.

2. Fundamentação Teórica

Este capítulo tem o objetivo de fornecer os conceitos necessários para o entendimento deste trabalho.

2.1 Kernel e Permissões Unix

O kernel é uma parte fundamental de um sistema operacional, é o componente responsável por intermediar a comunicação das aplicações com o hardware do computador [23]. Como o *kernel* do Android é o Linux, ele tem o mesmo sistema de permissões das distribuições Linux, que na verdade foi herdado de um antigo e famoso sistema operacional, o Unix.[10]

As permissões tradicionais do Unix são uma forma de gerenciar o acesso dos usuários a dados do sistema de arquivos, elas são utilizadas por diversos sistemas operacionais, incluindo o Android [22].

Com essas permissões, é possível criar um ambiente isolado de forma que apenas um usuário, ou um grupo específico, consiga interagir com determinados arquivos, e de fato, o Android se utiliza dessa característica para isolar seus aplicativos.

Dessa forma, um aplicativo pode salvar informações sensíveis no dispositivo (por exemplo, dados financeiros), sem se preocupar com outros aplicativos instalados que podem ser perigosos (um *malware* disfarçado de um jogo, por exemplo).

No entanto, outra característica que o Android obtém ao utilizar o kernel Linux é a existência do usuário root, que consegue burlar o sistema de permissões.

2.2 Root

Root é o nome do superusuário de diversos sistemas operacionais, incluindo o Android. O termo "*root*" também pode se referir ao acesso *root*, que é quando os privilégios do usuário root são obtidos, normalmente de forma temporária apenas para realizar uma determinada ação, como por exemplo instalar um programa.[11]

O Android foi projetado de forma que o *root* não seja necessário para o uso comum dos seus usuários, por isso, o sistema por padrão não oferece um meio para que as pessoas consigam o acesso *root* [13].

Contudo, muitos usuários optam por *rootear* seus aparelhos [4], sem refletir que esse excesso de poderes pode quebrar o isolamento do sistema, pois os

[10] Disponível em <https://www.opengroup.org/membership/forums/platform/unix>, acessado em 07 de setembro de 2020.
[11] Disponível em <https://www.ssh.com/iam/user/root/#what-is-a-root-user?>, acessado em 07 de setembro de 2020.

aplicativos que obtém privilégios de *root* conseguem acessar os dados de outros, sem sofrerem as restrições que normalmente seriam impostas pelo sistema de permissões [13].

2.3 Obtendo o Acesso Root no Android

Como detalhado em Nguyen-Vu et al. (2017), existem muitos caminhos para alcançar o acesso *root* no Android. Atualmente, a maneira mais genérica e popular envolve o *bootloader* e o *recovery* do dispositivo [13].

Bootloader é um programa que tem como função principal inicializar um sistema operacional.[12] Um exemplo deles é o grub,[13] que é utilizado por muitas distribuições Linux.

Em aparelhos Android, o *bootloader* também permite que arquivos de dados sejam instalados nas partições do dispositivo, ação conhecida pelo termo "*flashing*", contudo, normalmente ele só aceita arquivos assinados pelo fabricante [22].

Por sua vez, o *recovery* é um sistema existente nos aparelhos Android que fica armazenado em uma partição diferente do sistema principal. Ele tem funções de suporte, como instalar atualizações (por padrão somente de assinatura conhecida) e apagar partições de dados do usuário [20].

Então, o primeiro passo para obter o *root* é desbloquear o *bootloader*, isto é, fazer com que ele permita a instalação de arquivos não assinados. Esse é um procedimento que pode variar de acordo com o fabricante de cada aparelho, mas que normalmente envolve o uso da ferramenta fastboot,[14] que utiliza um protocolo de mesmo nome.[15]

Após esse desbloqueio, por meio do *bootloader*, é possível instalar um *recovery* personalizado que normalmente tem mais funções do que o original e também não tem restrição em relação a assinatura. Uma dessas funções é a de instalar arquivos com instruções para modificar o Android, permitindo que novas funcionalidades sejam adicionadas [22].

Nesse ponto entra em ação o Magisk, que pode ser instalado por um *recovery* personalizado. Após esse procedimento surge no Android o aplicativo Magisk Manager, que é um gerenciador para o acesso *root*. Ele é ilustrado na Figura 2.

[12] Disponível em <https://source.android.com/devices/bootloader>, acessado em 07 de setembro de 2020.
[13] Disponível em <https://www.gnu.org/software/grub/>, acessado em 07 de setembro de 2020.
[14] Disponível em <https://source.android.com/setup/build/running#unlocking-the-bootloader>, acessado em 07 de setembro de 2020.
[15] Disponível em <https://android.googlesource.com/platform/system/core/+/refs/heads/master/fastboot/README.md>, acessado em 07 de setembro de 2020.

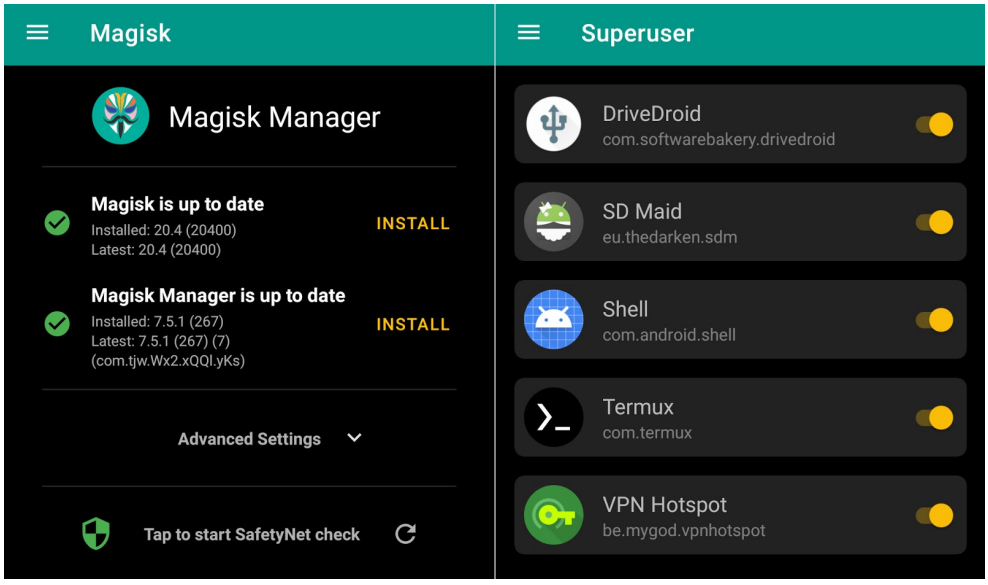


Figura 2: O aplicativo Magisk Manager.
Fonte: O autor.

Quando o Magisk está instalado, qualquer aplicativo do aparelho pode solicitar o acesso *root* de diversas formas. Uma delas é executando um binário de linha de comando chamado su.[16] Quando ele é executado no Android, surge um alerta na tela para que seja possível definir se determinado aplicativo receberá ou não o acesso privilegiado, como é mostrado na Figura 3.

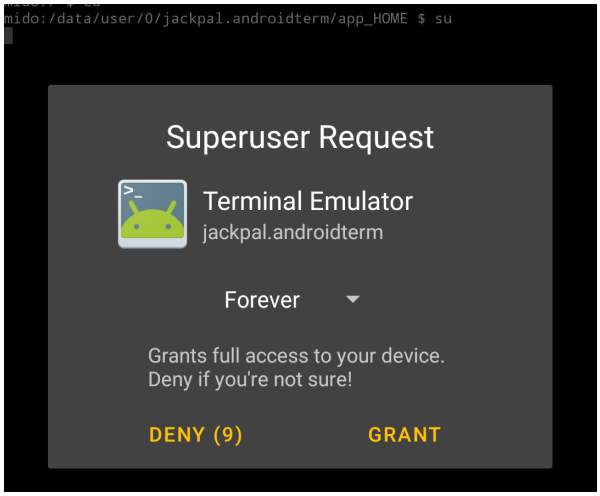


Figura 3: Solicitação de acesso *root* feita pelo aplicativo Terminal Emulator.
Fonte: O autor.

Uma outra forma de obter privilégios *root* é através do adb, abreviação de *Android Debug Bridge*, que é uma ferramenta com muitas funções que conecta computadores de desenvolvimento com dispositivos Android [11].

Adicionalmente, no Android existe um elemento conhecido como propriedade, que informa e determina algumas características do sistema. Dessa

[16] Disponível em <https://man7.org/linux/man-pages/man1/su.1.html>, acessado em 08 de setembro de 2020.

forma, há 3 propriedades “ro.debuggable”, “ro.secure” e “service.adb.root” que se configuradas com um determinado valor permitem o acesso *root* através do adb,[17] isto é, sem utilizar ferramentas de terceiros no aparelho, como o Magisk.

No entanto, para realizar a alteração dessas propriedades, sem o *root*, o usuário precisa instalar um sistema modificado que já venha com essas propriedades configuradas, o que acontece por exemplo em versões do Android que são utilizadas por emuladores [20].

Dessa forma, essa é uma forma mais específica de obter *root*, e portanto, ela não é tão popular quanto o Magisk.

2.4 Técnicas de Detecção de Root

Segundo Nguyen-Vu et al. (2017), que é o trabalho mais recente entre os que foram discutidos na Seção 1.4, existem 6 classificações para as técnicas de detecção do acesso *root* que são descritas a seguir.

2.4.1 Checagem de Aplicativos Instalados

O intuito desta verificação é investigar se determinadas aplicações, que possam indicar o acesso *root*, estão instaladas no dispositivo. Dado que existem aplicativos que só funcionam com *root* habilitado, como por exemplo o Titanium Backup,[18] a presença deles é um forte indicativo. A Figura 4 apresenta um trecho de código exemplificando essa verificação.

```
private boolean checkInstalledApplications() {
  PackageManager pm = getPackageManager();
  /* obtendo a lista de aplicativos instalados */
  List<ApplicationInfo> packages = pm.getInstalledApplications(0);
  for (ApplicationInfo info : packages) {
      /* procurando pelo Titanium Backup */
      if (info.packageName.equals("com.keramidas.TitaniumBackup")) {
          return true;
      }
  }
  return false;
}
```

Figura 4: Exemplo de uma checagem de aplicativos instalados.
Fonte: O autor.

[17] Disponível em <https://android.googlesource.com/platform/system/core/+/master/adb/daemon/main.cpp#65>. Acessado em 23 de setembro de 2020.
[18] Disponível em <https://play.google.com/store/apps/details?id=com.keramidas.TitaniumBackup>. Acessado em 21 de setembro de 2020.

2.4.2 Checagem de Arquivos

Essa checagem consiste em procurar por diretórios no aparelho que indicam a presença do *root*. Um exemplo dessa técnica aconteceu no Pokémon GO, que começou a procurar por diretórios do TWRP e do Titanium Backup [10]. A Figura 5 ilustra uma busca por dois arquivos que indicam a presença do Magisk.

```
private boolean checkFiles() {
  PackageManager pm = getPackageManager();
  /* lista de arquivos que indicam root */
  String[] rootFiles = new String[] {
     "/system/media/theme/miui_mod_icons/com.topjohnwu.magisk.png",
     "/system/addon.d/99-magisk.sh"
  };
  for (String filename : rootFiles) {
      File file = new File(filename);
      if (file.exists()) {
          /* encontrou algum dos arquivos */
          return true;
      }
  }
  return false;
}
```

Figura 5: Exemplo de uma busca por arquivos do Magisk.
Fonte: O autor.

2.4.3 Checagem de Processos, Serviços e Tarefas

Ainda no trabalho de Nguyen-Vu et al. (2017), os autores haviam mencionado que a partir do Android 5 não seria mais possível para os aplicativos obterem tarefas que não fossem deles mesmos. Além disso, nas versões mais recentes do Android, também não é mais permitido que um aplicativo liste os processos ou serviços[19] de outros de forma simples e sem restrições como acontecia no passado,[20] então esse tipo de checagem se tornou inviável.

2.4.4 Checagem de Execução de Comandos

Este é um tipo de verificação muito abrangente e que possui diversas possibilidades, pode-se tentar desde a execução do su, que permite a elevação de privilégios, até uma checagem de busca por arquivos. A Figura 6 demonstra um trecho de código que avalia se o comando su está disponível.

[19] Disponível em <https://developer.android.com/reference/android/app/ActivityManager#getRunningServices(int)>. Acessado em 22 de setembro de 2020.
[20] Disponível em <https://jaredrummler.com/2017/09/13/android-processes/>. Acessado em 22 de setembro de 2020.

```
private boolean checkCommandExecution() {
  try {
      /* se o comando nao existir, ocorre uma excecao */
      Process rootProcess = Runtime.getRuntime().exec("su");
      return true;
  } catch (Exception e) {
      return false;
  }
}
```

Figura 6: Exemplo de uma checagem que tenta executar o su.
Fonte: O autor.

2.4.5 Checagem de Propriedades do Sistema

Neste ponto o objetivo é buscar por propriedades indicando que a aplicação está rodando em um aparelho *rooteado*. Para este fim, uma opção é verificar os valores de “ro.debuggable”, “ro.secure” e “service.adb.root” que podem permitir o acesso *root* através do adb, como explicado na Seção 2.3. Além disso, até o momento, não existe uma forma de ler propriedades sem realizar uma execução de comandos,[21] portanto, essa checagem poderia ser considerada um subtipo da anterior. A Figura 7 apresenta um exemplo de implementação.

```
private boolean checkSystemProperties() {
   try {
       Runtime run = Runtime.getRuntime();
       /* obtendo o valor das propriedades com o comando getprop */
       Process debuggableProp = run.exec("getprop ro.debuggable");
       int debuggableValue = debuggableProp.getInputStream().read();
       Process secureProp = run.exec("getprop ro.secure");
       int secureValue = secureProp.getInputStream().read();
       Process srvRootProp = run.exec("getprop service.adb.root");
       int srvRootValue = srvRootProp.getInputStream().read();
       /* retorna se o adb consegue fornecer acesso root */
       return debuggableValue == '1'
           && (secureValue == '0' || srvRootValue == '1');
   } catch (Exception e) {  }
   return false;
}
```

Figura 7: Exemplo de uma checagem de algumas propriedades do sistema.
Fonte: O autor.

Essa checagem, no entanto, não é efetiva contra o Magisk, como é explicado na Seção 2.6.1.

[21] Disponível em <https://android.googlesource.com/platform/frameworks/base/+/master/core/java/android/os/SystemProperties.java#52>. Acessado em 23 de setembro de 2020.

2.4.6 Checagem de Permissões dos Diretórios

Este tipo de checagem existia porque segundo Sun et al. (2015), existem formas de obtenção de *root* que poderiam alterar as permissões de leitura e escrita de alguns diretórios. No entanto, tais mudanças não ocorrem com o Magisk e, portanto, essa checagem não é útil para detectá-lo [18].

Como é possível observar, a implementação é um fator muito importante para alcançar um resultado efetivo, visto que é necessário saber o que exatamente procurar com cada uma das técnicas.

2.5 SafetyNet Attestation

SafetyNet é um conjunto de serviços e APIs do Google que tem o objetivo de fornecer determinadas soluções de segurança para aplicativos do Android. Uma dessas APIs é a *SafetyNet Attestation* que pode ser utilizada por desenvolvedores para avaliar a integridade de um aparelho que estará executando seu aplicativo [27].

Para cumprir seu propósito, a API em questão fornece dois importantes parâmetros, *basicIntegrity* e *ctsProfileMatch*, ambos com valores booleanos. A Figura 8 demonstra parte de uma resposta de avaliação, solicitada pelo aplicativo "*com.example.myapplication*".

```
{
  "apkPackageName": "com.example.myapplication",
  "ctsProfileMatch": true,
  "basicIntegrity": true,
  "evaluationType": "BASIC",
  ...
}
```

Figura 8: Exemplo de uma resposta da API *SafetyNet Attestation*.
Fonte: O autor.

O parâmetro *basicIntegrity* informa o resultado de uma avaliação que, dentre outras coisas, falha caso detecte que o aparelho tem acesso *root* ou que a execução está ocorrendo em um emulador. Por sua vez, o parâmetro *ctsProfileMatch* entrega o resultado de uma avaliação que, dentre outras coisas, falha quando o aparelho tem um *bootloader* desbloqueado, usa uma ROM customizada (um Android modificado) ou falhou na avaliação do *basicIntegrity* [27].

Como a API considera o acesso *root*, ela é uma opção adotada por alguns aplicativos, como o Pokémon GO,[22] para dificultar a sua execução em aparelhos *rooteados*. Ademais, ela é gratuita e relativamente simples de utilizar, como pode ser visto no trecho de código da Figura 9.

[22] Disponível em <https://www.xda-developers.com/latest-update-to-pokemon-go>. Acessado em 27 de setembro de 2020.

```
private void sendSafetyNetRequest(String apiKey) {
  SafetyNetClient sfClient = SafetyNet.getClient(this);
  sfClient.attest(getRequestNonce(), apiKey).addOnSuccessListener(this,
    new OnSuccessListener<SafetyNetApi.AttestationResponse>() {
      public void onSuccess(SafetyNetApi.AttestationResponse response){
        /* recebendo resposta da API e enviando-a
           para ser checada no servidor da aplicacao */
        sendAttestationResponse(response.getJwsResult());
      }
    }).addOnFailureListener(this, new OnFailureListener() {
    public void onFailure(@NonNull Exception e) {
      /* tratar falhas de comunicacao */
    }
  });
}
```

Figura 9: Código que faz uso da API *SafetyNet Attestation*.
Fonte: Google.[23]

No entanto, há também alguns problemas referentes ao uso dessa API. O primeiro é que, em um aparelho *rooteado* com o Magisk, ela não detecta nem o acesso *root* e nem o *bootloader* desbloqueado (que, como foi explicado na Seção 2.3, é necessário para instalar o Magisk). Isto ocorre mesmo que, atualmente, exista uma previsão do próprio criador do Magisk, John Wu, de que no futuro não será mais possível esconder para essa API que um *bootloader* está desbloqueado.[24]

Outro problema é que o próprio Google não recomenda usar a API apenas para a verificação de acesso *root*, visto que além dela levar em conta outros fatores, atacantes reais podem passar no teste (se usarem o Magisk por exemplo) e usuários legítimos poderiam falhar nele [27].

Finalmente, essa API depende do Google Play Services para obter os dados que irão ser avaliados, assim, é necessário que ele esteja instalado no aparelho do usuário, o que pode não ser comum para pessoas que vivem em países mais restritivos como a China.[25]

2.6 Métodos de Evasão

Nesta seção serão explicados métodos de evasão, ou seja, que podem ser utilizados para esconder o acesso *root*.

[23] Disponível em <https://github.com/googlesamples/android-play-safetynet/>. Acessado em 27 de setembro de 2020.
[24] Disponível em <https://twitter.com/topjohnwu/status/1237830559365136384>. Acessado em 27 de setembro de 2020.
[25] Disponível em <https://www.theverge.com/2020/3/26/21194803/android-google-play-services-huawei-china>. Acessado em 27 de setembro de 2020.

2.6.1 Magisk

Magisk é o nome de um conjunto de ferramentas (do inglês *toolkit*) de código aberto, criado em 2016 por John Wu. Inicialmente, ele era apenas uma alternativa, ao SuperSU, para aqueles que desejavam *rootear* seus dispositivos.

Com o passar do tempo e a descontinuação do SuperSU [28], o Magisk se tornou não só a maneira mais popular para obter *root*, mas também para escondê-lo (como mostram os resultados do *survey* apresentado no Capítulo 3). A ferramenta, atualmente, já possui mais de 72 milhões de *downloads* [18].

Ao instalar o Magisk (como detalhado na Seção 2.3), visualmente parece que apenas houve a adição de mais um aplicativo, o Magisk Manager. Contudo, enquanto para os usuários basta abrir o Magisk Manager e, a partir de uma lista dos aplicativos instalados, marcar quais não devem encontrar o *root* (como mostra a Figura 10), o que acontece internamente é um pouco mais complexo.

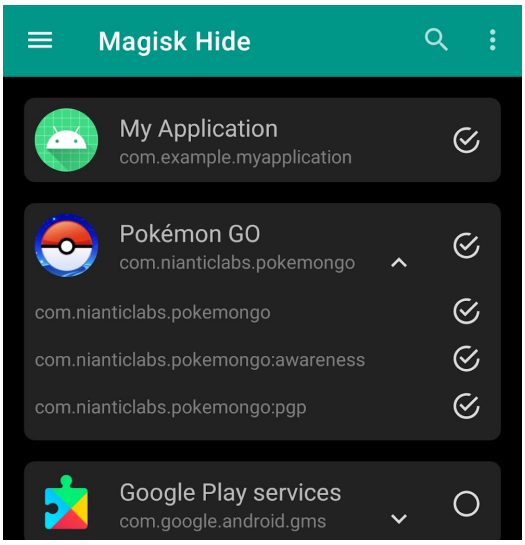


Figura 10: A tela do Magisk Manager que utiliza o MagiskHide.
Fonte: O autor.

O Magisk Manager atua como uma interface gráfica para outros programas que também são instalados. Um deles é o MagiskHide, responsável por esconder o acesso *root*, ele é escrito em C++ e desenvolvido como uma ferramenta de linha de comando [18].

Para entender o funcionamento do MagiskHide, é preciso que três características do Android sejam apresentadas. A primeira é o fato de que, a fim de melhorar desempenho e economizar memória, o Android tem um processo que fica em constante espera chamado de *Zygote*, que pré-carrega algumas das classes e recursos que os aplicativos costumam precisar. Com isso, quando um aplicativo vai ser iniciado, o *Zygote* copia a si mesmo, para criar um novo processo, e depois carrega o aplicativo [22].

As duas características restantes vem do *kernel* Linux. Uma é o *ptrace*, uma funcionalidade que permite que alguns processos possam monitorar e modificar o estado interno de outros, o que normalmente é usado por *debuggers* [29]. A outra é a funcionalidade de *namespaces* que é utilizada por aplicações de conteinerização (como o Docker)[26] para separar recursos (arquivos, redes, processos), de forma que um determinado grupo de processos não consiga ver recursos de outros [30].

Ao ser iniciado, o MagiskHide altera algumas propriedades do Android para valores padrões de aparelhos sem *root* (assim torna-se improvável que as checagens abordadas na Seção 2.4.5 funcionem) [18].

Em seguida, o *ptrace* é utilizado para monitorar o *Zygote*, então, antes da inicialização de cada aplicativo, o MagiskHide checa se o mesmo está na lista de alvos que não devem encontrar o *root*. Quando um alvo é encontrado, o MagiskHide altera o *namespace* dele, de forma que alguns diretórios que contém ferramentas do Magisk (como o /sbin) se tornam inacessíveis apenas para o alvo [18].

Por fim, um outro recurso fornecido pelo Magisk para dificultar a detecção do acesso *root* é a opção de trocar os nomes do Magisk Manager. Dessa forma, ele recebe um *package name* aleatório, e permite ao usuário escolher um nome para o aplicativo, de forma que caso sejam listados os aplicativos instalados (como detalhado na seção 2.4.1), não é possível encontrar o termo “magisk”. A Figura 11 ilustra o Magisk Manager antes e depois dessa alteração.

Figura 11: Na esquerda o aplicativo original, na direita ele após uma troca de nomes.
Fonte: O autor.

2.6.2 Frida

Frida é o nome de um *toolkit* de código aberto que tem o objetivo de fazer o *hooking* das funções de uma aplicação em tempo de execução. Logo, a partir do Frida é possível modificar totalmente o comportamento de um programa [31]. Somado ao fato de que o Android é uma das diversas plataformas que são suportadas pelo Frida, é possível utilizá-lo para evadir detecções de acesso *root*.

Assim, duas ferramentas do Frida são comumente utilizadas, uma que atua como cliente (e é executada em um computador de desenvolvimento) e outra que desempenha o papel de servidor (e é executada na plataforma alvo, como um

[26] Disponível em <https://docs.docker.com/engine/security/security/>. Acessado em 29 de setembro de 2020.

aparelho Android). Dessa forma, o servidor injeta uma *engine* JavaScript nos alvos, e, posteriormente, recebe códigos do cliente, escritos em JavaScript, para controlar as instruções internas do programa [31].

A Figura 12 demonstra um exemplo de código JavaScript em que o método *checkCommandExecution* é alterado, de forma que independente da sua implementação original, seu valor de retorno será *false*.

```
Java.perform(function() {
  var mActivity = Java.use("com.example.myapplication.MainActivity");
  mActivity.checkCommandExecution.implementation = function() {
     console.log('[+] method hooked!');
     /* alterando a implementacao de checkCommandExecution
        para retornar false em todos os casos */
     return false;
  }
})
```

Figura 12: O *hooking* da detecção que foi ilustrada na Figura 6.
Fonte: O autor.

Como é possível notar, para utilizar o Frida é necessário que se conheça a estrutura interna da aplicação alvo (como nomes de classes, métodos, atributos), e visto que, criar um código para cada aplicação leva tempo e requer conhecimento técnico, surgiram formas mais fáceis de utilizar o Frida para propósitos específicos.

Uma delas é um código chamado fridantiroot[27] que tenta manipular várias funções comumente usadas na detectação do acesso *root*. Outra opção é a ferramenta *objection* [8], que internamente contém um código semelhante ao do fridantiroot, mas abstrai isso do usuário, necessitando apenas que o servidor do Frida esteja sendo executado no aparelho alvo.

2.6.3 Outros

Como mencionado na Seção 1.4, no passado houveram outros métodos de evasão que hoje não são muito efetivos ou já foram descontinuados (RootCloak, Hide My Root). Uma opção que tem baixa probabilidade de ficar desatualizada, e pode ser ainda mais eficaz que o Magisk, é o uso da engenharia reversa, a fim de adulterar funções de detecção de um aplicativo. Existem algumas ferramentas que podem ser utilizadas para esta finalidade, como o apktool.[28] No entanto, essa medida requer um tempo consideravelmente maior que as demais, já que é específica para cada aplicação, além de um grande conhecimento técnico. Portanto, este trabalho não detalha o uso da engenharia reversa como forma de evasão, mas aborda medidas de proteção contra ela (no Capítulo 5).

[27] Disponível em <https://codeshare.frida.re/@dzonerzy/fridantiroot/>. Acessado em 30 de setembro de 2020.
[28] Disponível em <https://github.com/iBotPeaches/Apktool>. Acessado em 30 de setembro de 2020.

3. Survey com Usuários e Desenvolvedores

Este capítulo descreve a metodologia e a análise dos resultados referente a uma pesquisa, realizada no formato de *survey*, para coletar informações de usuários e desenvolvedores do Android sobre técnicas de detecção de acesso *root*. A apresentação dos resultados do *survey* segue as sugestões de Kasunic (2005).

3.1 Motivação

O propósito da pesquisa foi entender quais eram as técnicas de detecção de acesso *root* mais utilizadas na prática. Nesta pesquisa, considerou-se tanto as detecções implementadas pelos desenvolvedores, quanto às formas de evasão utilizadas pelos usuários.

3.2 Procedimento

Através do serviço Google Forms foi gerada uma URL que fornecia acesso a um *survey* online. No entanto, havia uma preocupação quanto a dificuldade de encontrar pessoas que já tiveram experiência com a detecção de *root*, logo houve uma preferência por ambientes mais técnicos, onde a chance de encontrar desenvolvedores Android era maior. O *survey* foi então divulgado em *threads* de e-mail de faculdades, grupos de tecnologia em aplicativos de mensagens e fóruns de discussão sobre o Android (Reddit, Android Forum, CNET).

Além disso, a pesquisa tinha duas versões, uma em português (cujas perguntas são apresentadas no Apêndice A) e outra em inglês, que foram compartilhadas de acordo com o idioma principal de cada canal de comunicação. A versão em português foi divulgada no dia 24 de junho de 2020, já a versão em inglês no dia 02 de julho de 2020. O recebimento de respostas em ambas as versões foram encerradas em 31 de agosto de 2020.

3.3 Estrutura

Antes de iniciarem as perguntas principais do *survey*, o participante era apresentado a um termo de consentimento que detalhava a garantia de anonimato quanto a sua participação, o objetivo da pesquisa, os pesquisadores responsáveis por ela e seu público alvo (usuários e desenvolvedores Android).

Ao aceitar o termo, o participante deveria responder duas perguntas obrigatórias de resposta única. O objetivo delas era classificá-lo entre dois grupos de acordo com o seu perfil:

- Grupo 1: desenvolvedores que tinham experiência com detecção de *root;*

- Grupo 2: usuários que já tentaram esconder o acesso *root*.

Para o grupo 1, destinado a desenvolvedores que tinham experiência com detecção de *root*, a terceira pergunta era obrigatória e de múltipla escolha, seguida de cinco não obrigatórias e de resposta aberta.

Todos que não se enquadraram no primeiro grupo eram classificados para o grupo 2, então para eles a terceira pergunta era obrigatória e de resposta única, com o objetivo de selecionar apenas aqueles que já tentaram esconder o *root*. Para quem nunca o fez, o *survey* era encerrado. Para os que continuaram, eram realizadas mais três perguntas referentes a sua experiência, sendo a quarta obrigatória e de múltipla escolha e as outras duas não obrigatórias e de resposta aberta.

3.4 Participantes

O *survey* alcançou um total de 262 participantes enquanto esteve online por um período de aproximadamente dois meses. Sendo 151 na versão de língua portuguesa e 111 na versão de língua inglesa. A Tabela 2 demonstra os grupos nos quais os participantes foram divididos, ao longo do *survey*.

Classificações	Quantidade
Grupo 1 - Desenvolvedores que já detectaram o root	17
Grupo 2 - Usuários que já esconderam o root	106
Participantes que não se qualificaram para os grupos	139
Total de participantes	262

Tabela 2: Agrupamentos dos participantes que responderam o *survey*.
Fonte: O autor.

3.5 Resultados

O objetivo da pergunta 1 *(Você trabalha ou já trabalhou no desenvolvimento de aplicativos Android?*) foi entender o perfil da pessoa que estava respondendo o *survey* e, consequentemente, determinar as futuras perguntas, da terceira em diante.

É possível notar na Figura 13 que apenas 17 dos 116 desenvolvedores se classificaram para o grupo 1 (com experiência no acesso *root*), o que é aproximadamente 14%. O restante dos participantes (ou seja, alguns dos desenvolvedores e os usuários) foi classificado para o grupo 2.

A pergunta 2, cujos resultados são apresentados na Figura 14, abordou o tempo de experiência da pessoa com desenvolvimento Android. Considerando apenas aqueles que forneceram uma resposta positiva para a primeira pergunta, os desenvolvedores Android, aproximadamente 40% tinham menos de 6 meses de experiência e 60% menos de 1 ano.

Após essas duas perguntas referentes ao perfil do participante, as próximas variavam de acordo com o grupo que o participante havia sido classificado. Dessa forma, os resultados que serão apresentados a seguir se referem ao grupo 1.

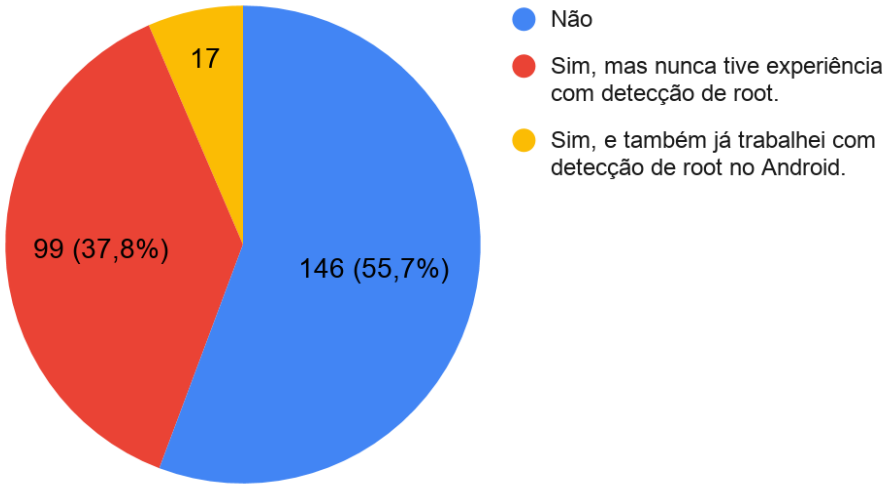


Figura 13: “Você trabalha ou já trabalhou no desenvolvimento de aplicativos Android?”
Fonte: O autor.

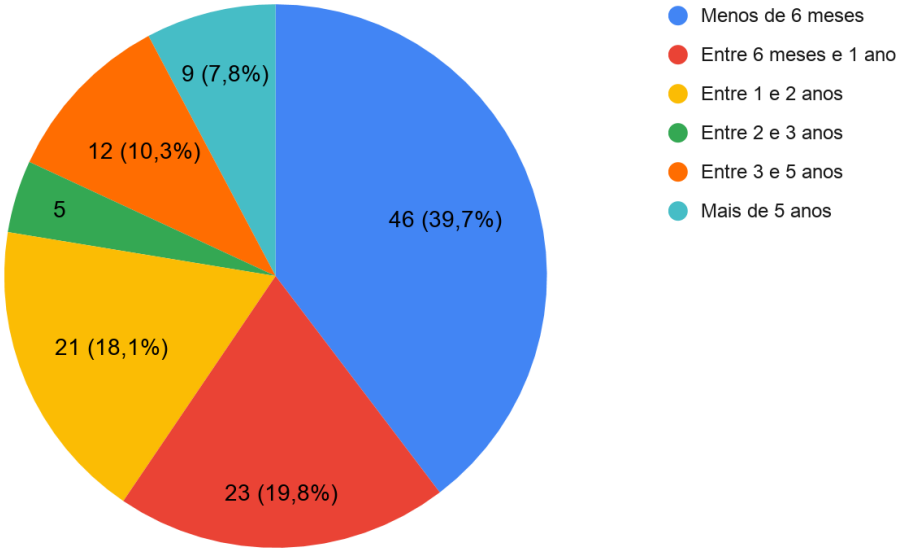


Figura 14: “Qual o seu tempo de experiência com desenvolvimento Android?”
Fonte: O autor.

3.5.1 Resultados do Grupo 1: Desenvolvedores com Experiência em Detecção de Root

A pergunta 3 investigou quais bibliotecas haviam sido utilizadas para realizar a detecção de *root*. Apesar de algumas terem sido apontadas, como demonstra a Figura 15, aproximadamente 58% das pessoas (10 dos 17) informaram que implementaram a detecção por conta própria.

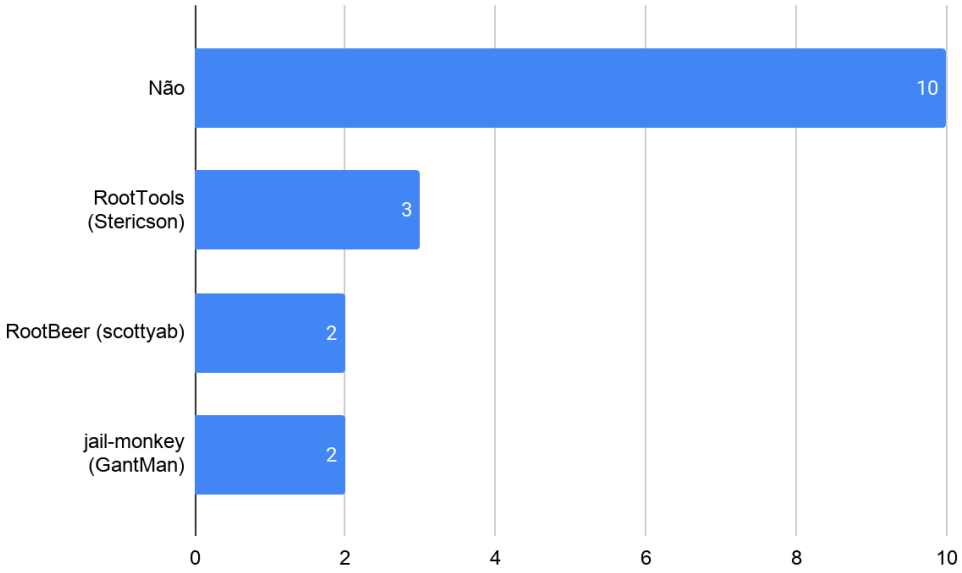


Figura 15: “Você utilizou alguma biblioteca para a detecção de root?”
Fonte: O autor.

A pergunta 4 foi “*Se você já implementou a detecção por conta própria, quais as principais verificações que eram realizadas*?”. Ela obteve um total de sete respostas, que determinaram as seguintes verificações:

- Busca por arquivos protegidos;
- Busca pelo binário su;
- Verificação de permissão de escrita em alguns diretórios;
- Tentativa de executar comandos para escalar privilégio;
- Verificação do parâmetro *ctsProfileMatch* da API *SafetyNet Attestation*.

As respostas obtidas são apresentadas abaixo.

“*Busca por arquivo protegidos no sistema, permissão de escrita em diretório protegidos, procura por sudo ou su, appId do closed*”
Participante 1, Grupo 1.

“*aplicativo de root checker*”
Participante 2, Grupo 1.

“Check $PATH for su command”
Participante 3, Grupo 1.

“Check cts”
Participante 4, Grupo 1.

“shell command ‘which su’, search for /su/, see if we can write a file to /system/”
Participante 5, Grupo 1.

“Just Installed root checker & checked”
Participante 6, Grupo 1.

“execute sudo linux command”
Participante 7, Grupo 1.

A pergunta 5 foi “*Quando você adiciona a detecção em um aplicativo, qual costuma ser o contexto que gera essa necessidade*?”. Nesse ponto, houve um total de sete respostas, informando que ao implementar a detecção, o objetivo foi reduzir a possibilidade de que usuários se aproveitem dos aplicativos, evitando ações maliciosas como fraudes em compras internas ou trapaças no contexto da aplicação. As respostas dos participantes são apresentadas a seguir.

“*Anti spoofing*”
Participante 1, Grupo 1.

“*aplicativos com possibilidade de hackear compras internas*”
Participante 2, Grupo 1.

“*Procurar indícios de localização simulada*”
Participante 3, Grupo 1.

“*The app needs root to work*”
Participante 4, Grupo 1.

“*To prevent users from cheating*”
Participante 5, Grupo 1.

“*There is no real need for this unless we are trying to make things easier, or we need to make modifications to the system and it's configuration files.*”
Participante 6, Grupo 1.

“*To take advantages of that app which are not allowed by manufacturer.*”
Participante 7, Grupo 1.

A pergunta 6 procurou entender "*qual é a melhor forma para um aplicativo realizar a detecção de acesso roo*t". As cinco pessoas que responderam essa pergunta forneceram opiniões razoavelmente diferentes. Três respostas (participantes 1 a 3) apontaram para alguma das técnicas mencionadas na pergunta 4, outra sugeriu que técnicas menos conhecidas seriam melhores (participante 4) e, por fim, o participante 5 disse que a melhor opção era utilizar a maior quantidade possível de técnicas.

"*Verificar por aplicativos conhecidos de root e arquivos, possivelmente incluindo uma solicitação de teste ao root.*"
Participante 1, Grupo 1.

"*Check cts*"
Participante 2, Grupo 1.

"*Looking for any modifications to the /system partition. Look for a new shell in /system/bin, execute "which su" / "whereis su" in a shell. Look for modifications to certain configuration files in /system/etc. Check to see if any partition has been remounted. Look for certain processes that may look strange.*"
Participante 3, Grupo 1.

"*adicionar também medidas de seguranças não muito famosas, pois não serão medidas conhecidas a serem burladas*"
Participante 4, Grupo 1.

"*Não acho q exista a melhor forma, pois acredito que a melhor é o conjunto de todas , mas se tivesse q dizer só uma acredito que as de mais baixo nível são mais eficientes contra script kids*"
Participante 5, Grupo 1.

A pergunta 7 foi "*Você acredita que as formas de detecção que você conhece poderiam ser burladas ou talvez gerar falsos-positivos? De que forma?*". Aqui quatro das seis respostas (participantes 1 a 4) apontaram o Magisk como uma possível forma de evasão sobre as técnicas, considerando que o participante 2 provavelmente cometeu um erro de digitação. Enquanto as outras respostas (participantes 5, 6) sugeriram modificar comandos do sistema para não entregarem informações úteis para a detecção ou utilizar um módulo do Xposed. Além disso, nenhuma resposta mencionou falsos-positivos.

"*Are you talking abt magisk? ??*"
Participante 1, Grupo 1.

“*O aplicativo Magister em certas ocasiões burlam detecções de root*”
Participante 2, Grupo 1.

“*Magisk has some hiding tools that counter-work the root detection*“
Participante 3, Grupo 1.

“*Using MagiskProp*”
Participante 4, Grupo 1.

“*If the application does not provide its own binaries. Or check to see if the binaries are legit. Then there could be a possibility for a bypass to have taken place. A binary "ls" command can be modified so anytime "ls /su " is run it returns an error.*”
Participante 5, Grupo 1.

“*Sim, existem diversas aplicações e scripts prontos que tentam burlar. O xposed por exemplo.*”
Participante 6, Grupo 1.

Por fim, a pergunta 8 foi a última para o grupo 1 e dizia “*Você gostaria de fornecer algum comentário adicional?*”. Nessa questão, houve um total de três respostas, dentre elas uma dizia que os aplicativos ainda tem muita dificuldade quando o assunto é detecção, enquanto outra sugeriu que nada está completamente escondido.

“*as respostas foram com base em usos não muito recente de root no android*”
Participante 1, Grupo 1.

“Os apps modernos ainda possuem muita dificuldade para detectar acessos ao root dos telefones visto que mesmo aplicativos como Netflix e BB que negariam acesso ao serviço, acabam liberando os mesmos quando a aplicação que fornece root está hidden ou disfarçada”
Participante 2, Grupo 1.

“This is a game of 1 ups and there's always ways around. Nothing is completely hidden”
Participante 3, Grupo 1.

A seguir serão apresentadas as perguntas que foram realizadas para os participantes do grupo 2.

3.5.2 Resultados do Grupo 2: Usuários com Experiência em Esconder o Root

A pergunta 3 tinha o objetivo de selecionar as pessoas com experiência tentando esconder o *root*. Conforme pode ser observado na Figura 16,

aproximadamente 43% delas informaram que já tiveram um aparelho com *root* e tentaram esconder isso de algum aplicativo.

O *survey* continuou apenas para aqueles que responderam positivamente para a pergunta 3. A pergunta 4 questionou quais métodos os participantes utilizavam para esconder o *root*. Aqui o Magisk liderou como a principal opção, sendo apontado por aproximadamente 90% das respostas, como é ilustrado na Figura 17.

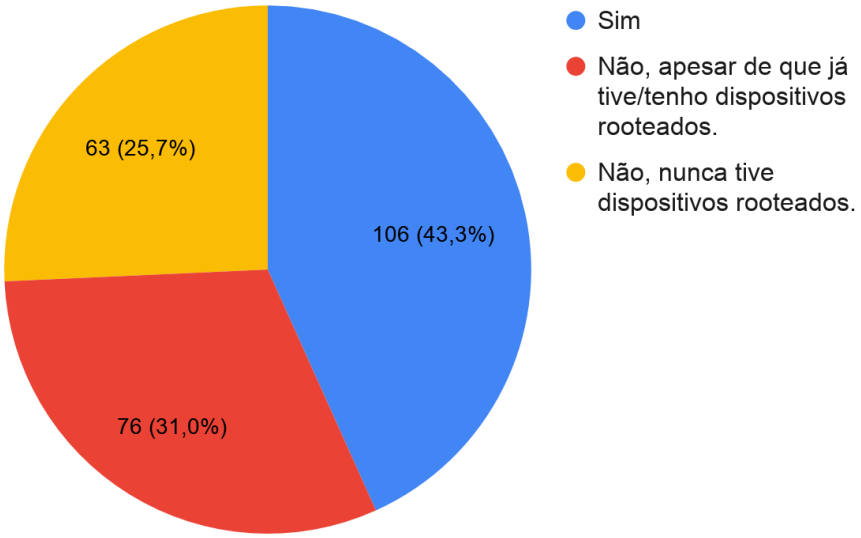


Figura 16: “Você já utilizou alguma técnica para esconder, para um aplicativo, o fato de que o dispositivo (ex: celular, tablet) tinha acesso root?”
Fonte: O autor.

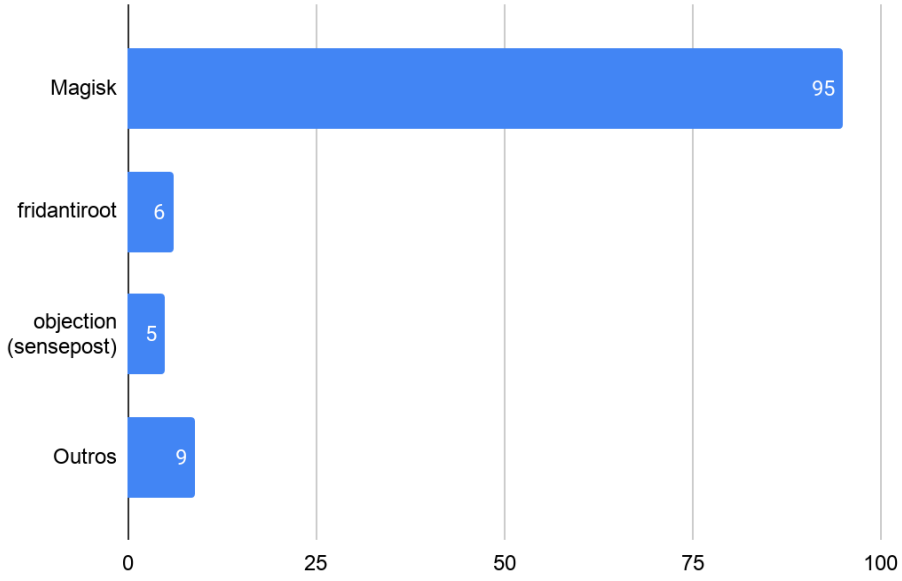


Figura 17: “Que mecanismo você já usou para esconder o acesso root? (ex: aplicativo, framework ou ferramentas envolvidas)”
Fonte: O autor.

A pergunta 5 buscou entender contra quais aplicativos as pessoas tentavam esconder o *root*. Ela recebeu 69 respostas, 50% delas apontaram para algum aplicativo financeiro (como bancos e carteiras digitais), seguido por Netflix e Snapchat com 13 e 12 menções, respectivamente, o que é demonstrado na Figura 18.

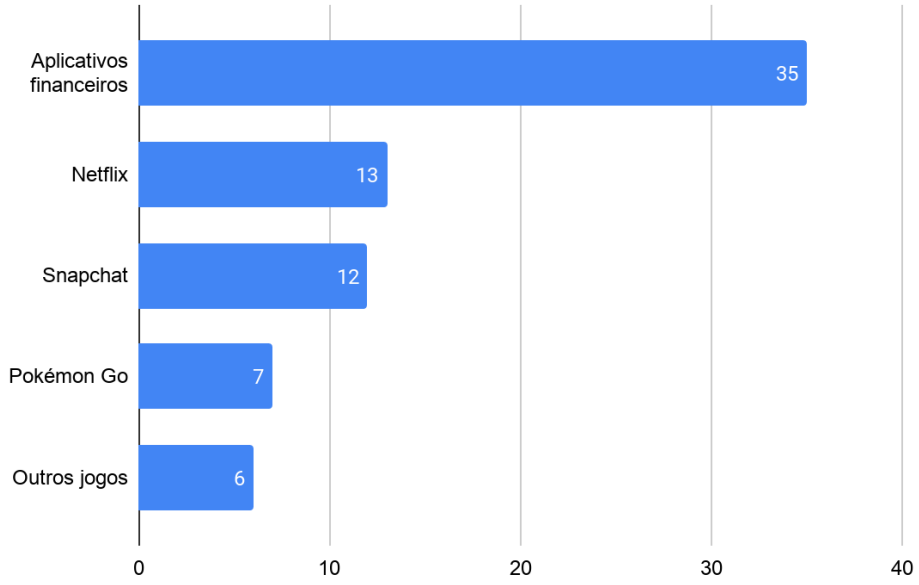


Figura 18: “Se possível, informe aplicativos contra os quais você conseguiu esconder o root, e se já houve algum para o qual você não conseguiu.”
Fonte: O autor.

Por fim, a pergunta 6 foi a última para o grupo 2 e dizia “*Você gostaria de fornecer algum comentário adicional?*”. Ela recebeu 19 respostas. Dentre elas, uma dizia que às vezes os aplicativos usam métodos novos para detectar o *root*. Enquanto outra disse que segurança é uma questão de camadas, então quanto mais dificuldades existirem para quem tentar esconder o *root*, menos o esforço valerá a pena, e portanto, menor será a chance da evasão ocorrer.

“*Segurança de aplicativos não é uma questão de ficar protegido e sim de escolher quem pode atacar. Por mais simples que seja a solução de detecção de root ela é válida, pois vai ‘filtrar’ por nível técnico os atacantes. Quanto mais sofisticada a solução menor será a quantidade de atacantes. Assumindo que sempre terá uma forma de contornar a segurança, essa ‘conta’ deve ser usada para tornar a exploração tão custosa (tempo de pesquisa por exemplo) que não valha a pena o esforço para o atacante/fraudador.*”

Participante 2, Grupo 2.

“*Some apps like Pokémon can sometime bypass android storage permmision and instead of looking for root, thees apps look for files of susspicious apps like magisk manager atc.....*”

Participante 16, Grupo 2.

3.6 Conclusões

Considerando os resultados do grupo 1 (desenvolvedores que tinham experiência com detecção de *root*), foi possível ver sinais de que mesmo com a mudança do SuperSU para o Magisk nos dias de hoje, a detecção continua muito menos madura do que a evasão.

Primeiramente, as respostas para a pergunta 1 mostraram que apenas 14% dos desenvolvedores Android que participaram da pesquisa, já haviam tido alguma experiência detectando o acesso *root*.

Em seguida, um indicativo da falta de padronização na detecção apareceu nas respostas para as perguntas 3 e 4 do grupo 1, quando 58% dos participantes disseram que implementaram a detecção por conta própria (talvez por desconhecimento ou insuficiência das técnicas existentes) e cada um utilizando uma verificação diferente.

Por último, as respostas das perguntas 6 e 7 do grupo 1 sugerem que não existe um consenso em relação a melhor forma de realizar a detecção e que o Magisk pode ser um problema para as técnicas conhecidas pelos desenvolvedores.

Em contraste com o grupo anterior, ao analisar os dados do grupo 2 (usuários que já tentaram esconder o acesso *root)* observa-se que a evasão parece dispor de métodos mais maduros e bem estabelecidos. É possível notar que o Magisk é a forma mais popular para esconder o *root*, apontado por quase 90% dos participantes do grupo 2 na pergunta 4.

Adicionalmente, é possível concluir que o foco daqueles que escondem o *root* são os aplicativos financeiros, talvez por também serem os que mais fazem uso da detecção. A pergunta 5 mostrou que o interesse em realizar a evasão neles foi mais do que o dobro do segundo lugar, o aplicativo da Netflix.

Por fim, vale ressaltar que dos 17 desenvolvedores classificados para o grupo 1, menos da metade respondeu as perguntas não obrigatórias. Essa quantidade não muito expressiva de respostas pode ter reduzido a precisão da compreensão do atual cenário da detecção.

4. Avaliação de Eficácia da Detecção

Os resultados do *survey* descritos no capítulo anterior demonstraram que as técnicas de detecção estão menos maduras do que as técnicas de evasão. Esses dados motivaram a necessidade de avaliar o quão eficazes estão as implementações das técnicas de detecção contra os métodos de evasão. Assim, serão apresentados dois estudos empíricos, o primeiro envolvendo aplicativos criados para realizar a detecção, enquanto o segundo envolve aplicativos mais baixados. Em ambos, o método de evasão utilizado foi o Magisk, visto que ele foi considerado o mais popular da atualidade (como demonstraram os resultados do Capítulo 3).

4.1 Análise de Aplicativos de Checagem de Root

Nesta seção será explicada uma análise que foi realizada com múltiplos aplicativos que proveem a funcionalidade de detecção de *root*.

4.1.1 Motivação

O objetivo desta análise foi avaliar o quão eficazes são as atuais implementações das técnicas de detecção. Dessa forma, foram escolhidos aplicativos criados com essa finalidade, na busca de alcançar as melhores implementações de identificação do *root*.

4.1.2 Ambiente da Análise

Como o dispositivo pode influenciar diretamente nos resultados obtidos, nesta análise foi escolhido um ambiente no qual o acesso *root* era mais difícil de ser detectado. O aparelho utilizado foi um Xiaomi Redmi K20 (*codename davinci*) que passava em ambos os testes da API *SafetyNet Attestation* (descrita na Seção 2.5), com *bootloader* desbloqueado, utilizando como *recovery* o TWRP na versão 3.4.0, a ROM *Pixel Experience* na versão 10.0-20200531-14-31 (Android 10) e *rooteado* por meio do Magisk na versão 20.4. Além disso, para dificultar a detecção, o aplicativo Magisk Manager teve seu nomes alterados (como demonstrado na Figura 11).

4.1.3 Procedimento

No final de junho de 2020, foi analisado um total de 136 aplicativos. Para escolhê-los foram salvos os 250 primeiros resultados retornados em buscas no Google Play, para cada um dos seguintes termos: “root detect”, “root check”, “root verity”, “magisk check”, “magisk detect”.

Essas 5 buscas totalizaram 1250 resultados, os aplicativos então foram filtrados obedecendo aos seguintes critérios:

- Todas as repetições foram removidas, cada aplicativo só poderia ser considerado apenas uma vez;
- O aplicativo deveria ter a detecção de *root* como uma de suas funções principais;
- Aplicativos pagos foram desconsiderados;
- Aplicativos não compatíveis com o aparelho de teste foram desconsiderados.

Após a aplicação dos filtros, 136 aplicativos foram selecionados para avaliação conforme apresentado no Apêndice B.

Para cada aplicativo selecionado, os seguintes passos foram realizados:

1) O aplicativo foi instalado no aparelho de teste através do Google Play;
2) O aplicativo foi adicionado na lista do MagiskHide e, em seguida, executado;
3) Foram observados os comportamentos do aplicativo (em busca de mensagens, imagens, alertas) a fim de verificar se ele conseguiu detectar o acesso *root* do aparelho;
4) Se fosse notado um comportamento que indicasse que o aplicativo está detectando o *root*, era realizada uma engenharia reversa no código do aplicativo para confirmar o resultado.

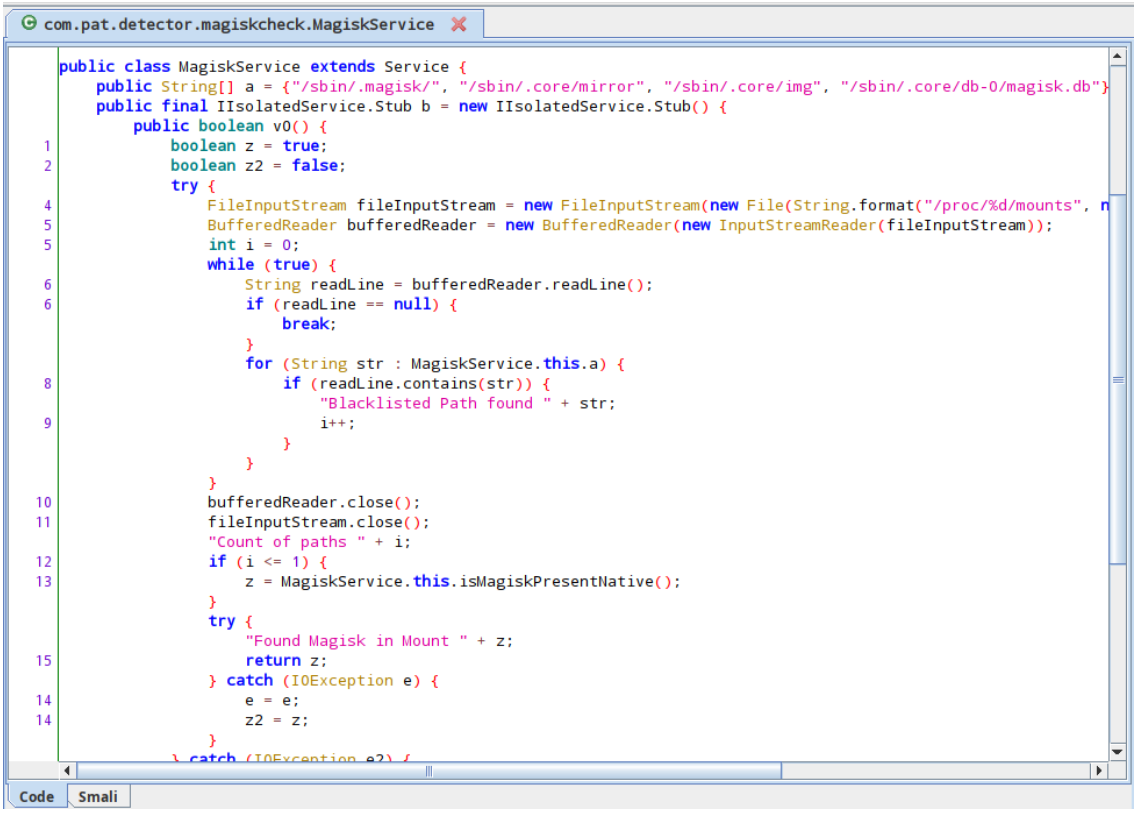
4.1.4 Resultados

De um total de 136 aplicativos analisados, apenas um conseguiu detectar o acesso *root* do aparelho, o aplicativo Magisk Detector (com o *package name* com.pat.detector, na versão 1.3).[29] Dessa forma, a detecção do Magisk ocorreu em menos de 1% das aplicações analisadas.

Ao realizar uma engenharia reversa no Magisk Detector, foi observado que ele se utiliza de uma falha pública[30] que afeta o Magisk desde a versão 19.[31] Por causa dessa falha, o Magisk não consegue esconder, para serviços isolados do Android, o caminho de alguns diretórios que são criados por ele. A Figura 19 ilustra um trecho de uma função do Magisk Detector, obtida por um programa que realiza engenharia reversa, mostrando que o código utilizado pelo aplicativo, na verdade,

[29] Disponível em <https://play.google.com/store/apps/details?id=com.pat.detector>. Acessado em 3 de outubro de 2020.
[30] Disponível em <https://darvincitech.wordpress.com/2019/11/04/detecting-magisk-hide/>. Acessado em 7 de outubro de 2020.
[31] Disponível em <https://medium.com/csg-govtech/diving-down-the-magisk-rabbit-hole-aaf88a8c2de0>. Acessado em 10 de outubro de 2020.

foi criado pelo responsável por divulgar a falha do Magisk mencionada anteriormente.[32]




```
public class MagiskService extends Service {
    public String[] a = {"/sbin/.magisk/", "/sbin/.core/mirror", "/sbin/.core/img", "/sbin/.core/db-0/magisk.db"};
    public final IIsolatedService.Stub b = new IIsolatedService.Stub() {
        public boolean v0() {
            boolean z = true;
            boolean z2 = false;
            try {
                FileInputStream fileInputStream = new FileInputStream(new File(String.format("/proc/%d/mounts", n
                BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(fileInputStream));
                int i = 0;
                while (true) {
                    String readLine = bufferedReader.readLine();
                    if (readLine == null) {
                        break;
                    }
                    for (String str : MagiskService.this.a) {
                        if (readLine.contains(str)) {
                            "Blacklisted Path found " + str;
                            i++;
                        }
                    }
                }
                bufferedReader.close();
                fileInputStream.close();
                "Count of paths " + i;
                if (i <= 1) {
                    z = MagiskService.this.isMagiskPresentNative();
                }
                try {
                    "Found Magisk in Mount " + z;
                    return z;
                } catch (IOException e) {
                    e = e;
                    z2 = z;
                }
            } catch (IOException e2) {
```

Figura 19: Código utilizado pelo Magisk Detector para procurar o Magisk.
Fonte: jadx.[33]

4.1.5 Considerações

Foi possível observar que ao analisar apenas aplicativos com foco na detecção, que deveriam ser aqueles com boas implementações para detecção de *root*, a taxa de eficácia contra o Magisk foi muito baixa.

Além disso, é interessante notar que mesmo existindo uma falha pública no Magisk que permite identificá-lo nas suas versões mais recentes, apenas um aplicativo se utilizava dela. Esses resultados reforçam as conclusões obtidas no *survey* (Seção 3.6) de que, atualmente, a detecção ainda é muito menos madura do que a evasão.

4.2 Análise dos Aplicativos Mais Baixados

Nesta seção, será apresentada uma análise que foi realizada nos aplicativos mais baixados do Google Play.

[32] Disponível em <https://github.com/darvincisec/DetectMagiskHide/blob/master/app/src/main/java/com/darvin/security/IsolatedService.java>. Acessado em 5 de outubro de 2020.
[33] Disponível em <https://github.com/skylot/jadx/>. Acessado em 5 de outubro de 2020.

4.2.1 Motivação

O objetivo desta análise foi avaliar quais os tipos de aplicativos que mais utilizam a detecção de *root*, como também avaliar o quão eficazes são as atuais implementações de detecção em aplicativos que são utilizados mais regularmente por usuários do Android. Por isso, foram escolhidos aplicativos populares de diversas categorias.

4.2.2 Ambiente da Análise

Como um dos propósitos desta análise era verificar se um aplicativo tentaria de alguma forma procurar pelo *root*, foi escolhido um ambiente que não provia métodos para esconder o *root*.

O aparelho utilizado inicialmente para esta análise foi um Moto G4 Plus (*codename athene*) que falhava em ambos os testes da API *SafetyNet Attestation*, com *bootloader* desbloqueado, utilizando o TWRP na versão 3.2.1, a ROM *Resurrection Remix* na versão 6.2.0-20180916 (Android 8.1.0) e com um acesso *root* fornecido pelo seu próprio sistema Android modificado. Para os aplicativos que aparentemente detectavam o *root* desse dispositivo, o ambiente apresentado na Seção 4.1.2 também era utilizado.

4.2.3 Procedimento

Em julho de 2020, o site AndroidRank[34] foi consultado para a obtenção de uma lista dos 20 aplicativos mais baixados de cada uma das 49 categorias do Google Play, resultando em um total de 980 aplicativos.

Todos esses aplicativos eram gratuitos, no entanto, o Google Play não permitiu que 127 deles fossem instalados por restrições de localidade ou, em alguns casos, por incompatibilidade com o aparelho de teste.

Para os 853 aplicativos restantes, os passos a seguir foram realizados:

1) Os aplicativos foram instalados pelo Google Play e, em seguida, executados;
2) Foram buscados comportamentos visuais que indicassem a detecção do *root* (alerta, imagens, erros) para aplicativos que precisavam de autenticação para liberarem funções, apenas a parte não autenticada deles foi considerada;
3) Todos aplicativos que não apresentavam sinais de detecção e tinham alguma funcionalidade de autenticação eram registrados;
4) Caso fosse considerado que um aplicativo estava detectando o *root*, ele era executado contra o Magisk no mesmo ambiente da análise anterior (detalhado na Seção 4.1.2).

[34] Disponível em <https://www.androidrank.org/>. Acessado em 19 de julho de 2020.

A avaliação de todos os aplicativos levou aproximadamente 6 semanas, e um *script* foi desenvolvido para auxiliar no processo.[35] As categorias e os aplicativos analisados estão listados no Apêndice C.

4.2.4 Resultados

Em 13 aplicativos foi possível confirmar a existência de alguma forma de detecção, o que significa que ela estava presente em aproximadamente 1.5% de um total de 853 aplicativos analisados. A Tabela 3 detalha os 13 aplicativos identificados, já a Figura 20 ilustra um exemplo de sinal de detecção de *root*.

Categoria	**Aplicativos**	**Quantidade**
Auto and Vehicles	com.gigigo.ipirangaconectcar	1
Beauty	kr.co.company.hwahae	1
Business	us.zoom.videomeetings	1
Finance	br.gov.caixa.tem, com.mobikwik_new, com.sbi.SBIFreedomPlus, com.sbi.lotusintouch	4
Food and Drink	com.mcdonalds.mobileapp	1
Game Adventure	com.nianticlabs.pokemongo	1
Health and Fitness	nic.goi.aarogyasetu	1
Parenting	com.hp.pregnancy.lite	1
Social	com.snapchat.android	1
Tools	com.google.android.gms	1

Tabela 3: Aplicativos que tinham detecção de *root*.
Fonte: O autor.

[35] Disponível em <https://github.com/dr01dr4v3/most-downloaded-apps>. Acessado em 07 de novembro de 2020.

Figura 20: Exemplo de sinal de detecção de *root*.
Fonte: O autor.

É importante observar que a categoria *Finance* liderou como aquela com a maior quantidade de aplicativos buscando o acesso *root*, como demonstrado na Tabela 3. A liderança dos aplicativos financeiros continuou mesmo ao se considerar a proporção de aplicativos analisados em cada categoria, que podia ser diferente por causa de alguns impedimentos de instalação provocados pelo Google Play.

A Figura 21 mostra a porcentagem de aplicativos que detectam o *root* para cada categoria que englobava algum dos 13 aplicativos identificados.

Além disso, todos os 13 aplicativos foram instalados no dispositivo de teste da Seção 4.1.2, adicionados na lista do MagiskHide e executados, na tentativa de analisar se eles ainda conseguiam encontrar o *root*, contudo todos falharam em detectá-lo.

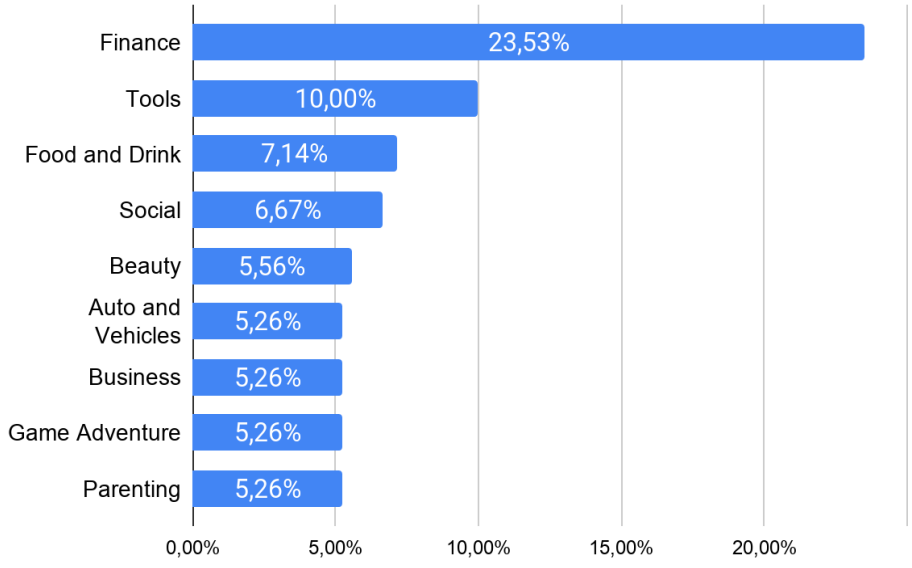


Figura 21: Percentual de aplicativos com detecção de *root* por categoria.
Fonte: O autor.

Finalmente é interessante notar que 374 aplicativos, dos 840 que aparentemente não procuravam pelo *root*, tinham uma funcionalidade que envolvia

a manipulação de dados sensíveis do usuário, ao solicitar credenciais de acesso e enviá-las pela internet.

4.2.5 Considerações

Observa-se que, atualmente, a detecção de *root* está concentrada no nicho dos aplicativos financeiros, o que condiz com o maior interesse dos usuários em esconderem o *root* para eles, que é demonstrado nos resultados do Capítulo 3. Uma das razões pode ser porque essa é uma área que tende a se preocupar mais com a segurança do que a maioria dado que, diferentemente de um aplicativo de calculadora ou um de consultar o clima, aplicativos da área de finanças envolvem dados privados ou serviços que podem ser fraudados.

Outros motivos que justificam o interesse de aplicativos financeiros em detectar se o aparelho tem acesso *root* seria impedir um usuário com um aparelho *rooteado* de usar a aplicação para reduzir os riscos de fraudes no serviço e de exposição dos dados do cliente. Existem casos de aplicativos que apenas coletam se o acesso *root* está habilitado, às vezes alertam o usuário, e o permitem continuar normalmente. Para esses aplicativos, um motivo possível é poder se eximir de culpa caso exista algum problema com determinado usuário no futuro.

Além disso, a taxa de aplicativos que buscavam pelo *root* se mostrou muito baixa, com aproximadamente 80% das categorias (39 de 49) sem ao menos um aplicativo com detecção. Estes resultados contrastam com o alto número de aplicativos que estavam trabalhando com informações sensíveis dos usuários. Sendo assim, é possível concluir que existe uma baixa preocupação dos desenvolvedores com a detecção de *root*, aumentando assim o risco de exposição dos dados dos clientes e de fraudes na aplicação (como detalhado na Seção 1.2).

Ademais, novamente nota-se o quão eficaz o Magisk é em esconder o *root*, ao ter alcançado sucesso em fazer isso contra todos os aplicativos analisados. Por fim, vale ressaltar uma limitação do estudo empírico apresentado. É possível que alguns dos aplicativos analisados estivessem buscando o *root* de forma silenciosa (sem nenhum tipo de mensagem visual) ou apenas após uma autenticação, nesses casos, os aplicativos acabaram sendo classificados como sem detecção.

5. Fortalecendo a Detecção de root

No capítulo anterior foram apresentados os resultados de dois estudos empíricos relacionados a detecção do acesso *root*. Seus resultados motivaram a apresentação de aprimoramentos, para que os desenvolvedores possam alcançar um melhor resultado na detecção de *root*. Uma das melhorias é a descoberta de ao menos um arquivo que expõe o Magisk, enquanto o outro é uma forma de checagem de aplicativos diferente da habitual, ambos são apresentados neste capítulo.

5.1 Técnicas Gerais

É importante ressaltar que todos os pontos apresentados neste capítulo podem ser evadidos de alguma maneira, visto que até o momento não é conhecida uma solução perfeita para detectar o *root*. Dessa forma, a meta é adicionar barreiras que dificultem a realização da evasão das técnicas implementadas.

Uma primeira barreira recomendada é a ofuscação de código. Por meio desta, é possível tornar o processo de engenharia reversa tão complicado e exaustivo que ele se torna inviável para quase todos os atacantes. Para o Android, existem tanto soluções pagas quanto gratuitas para realizar a ofuscação de código [32].

Outra barreira interessante é a utilização de uma biblioteca consolidada, como a RootBeer.[36] Apesar dela ser facilmente burlada pelo Magisk, ela é uma ótima opção caso o objetivo da detecção seja apenas alertar sobre possíveis riscos para o usuário.

A terceira barreira, se aplicável, é o uso da API *SafetyNet Attestation* (descrita na Seção 2.5). Para aplicativos que funcionam apenas de forma online, ainda é possível forçar que uma resposta assinada pela API tenha que ser enviada para o servidor da aplicação, a fim de liberar o acesso a determinada funcionalidade. Contudo, para decidir sobre a sua utilização, é importante considerar o contexto de cada aplicativo isoladamente, visto que ao depender da API do Google, o aplicativo poderá não funcionar para alguns usuários legítimos, como moradores da China.

5.2 Técnicas Específicas Contra o Magisk

Como foi visto nos resultados dos estudos empíricos realizados no Capítulo 4, o Magisk é extremamente eficaz contra os aplicativos atuais, mesmo já existindo uma falha nele que permite identificá-lo. Contudo, essa falha só afeta as versões de 19 em diante e, portanto, pode ser evitada pelos usuários. Dessa forma, a seguir

[36] Disponível em <https://github.com/scottyab/rootbeer>. Acessado em 04 de outubro de 2020.

serão apresentadas duas novas abordagens que conseguem identificar o Magisk, uma que aprimora uma técnica de detecção já existente e outra que pode ser considerada como uma forma de detecção inédita.

5.2.1 Busca por Nome e Conteúdo de Arquivos

O propósito deste ponto é demonstrar um procedimento que pode ser realizado para encontrar arquivos a serem verificados na checagem de arquivos (descrita na Seção 2.4.2) e, além disso, mostrar um arquivo que é criado pelo próprio Magisk e permite detectá-lo.

Como detalhado na Seção 2.6.1, o Magisk usa a funcionalidade de *namespaces* para esconder diretórios que são criados por ele no dispositivo. Contudo, essa solução não é perfeita, pois ainda existe a possibilidade de que um arquivo seja criado fora desses diretórios protegidos.

Portanto, a fim de realizar uma busca por arquivos expostos em outros diretórios, os passos abaixo foram reproduzidos no ambiente de teste detalhado na Seção 4.1.2:

1) Um aplicativo de terminal (sem a permissão de acesso ao armazenamento externo) foi instalado no dispositivo de teste, visto que assim seria possível procurar sinais do Magisk do ponto de vista de um aplicativo padrão;
2) O aplicativo de terminal foi colocado na lista do MagiskHide;
3) Foi procurado por arquivos que tinham o termo “magisk” em seus nomes ou conteúdos, dentro de diretórios comuns do Android e acessíveis pelo aplicativo (como /system, /vendor, /mnt).

Dessa forma, foi possível encontrar arquivos com esse nome ou conteúdo utilizando as ferramentas find e grep como é demonstrado na Figura 22.

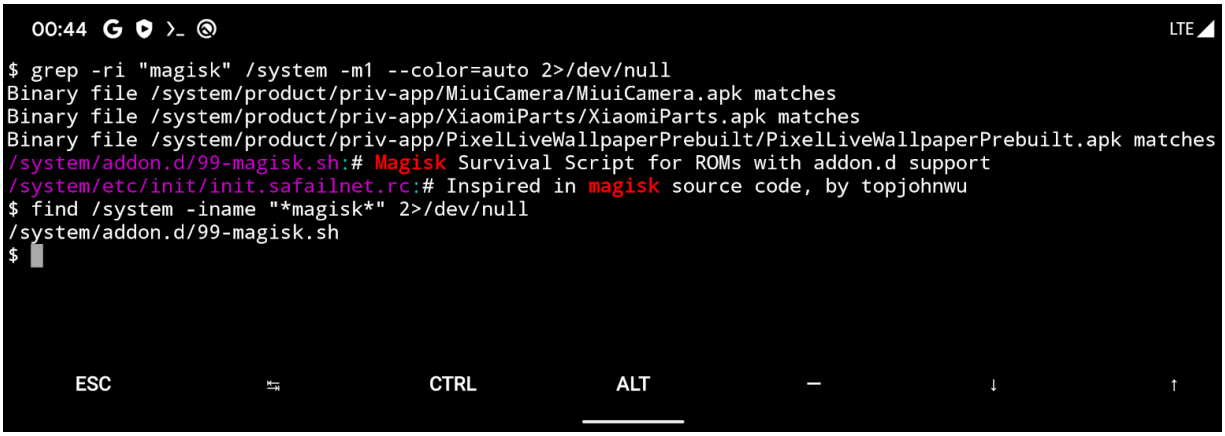


Figura 22: Buscando por arquivos relacionados ao Magisk no diretório /system.
Fonte: O autor.

Nesse ponto alguns cuidados devem ser tomados para evitar falsos-positivos, visto que nem todos os arquivos encontrados de fato eram indicadores de que o Magisk foi instalado no aparelho.

Entre os arquivos que aparecem na Figura 22, apenas "/system/addon.d/99-magisk.sh" realmente pode ser usado como um indicador. Esse arquivo é criado no momento da instalação do Magisk, caso o diretório "/system/addon.d" exista, o que pode ser verificado em um trecho de código do Magisk apresentado na Figura 23.

```
78  # addon.d
79  if [ -d /system/addon.d ]; then
80    ui_print "- Adding addon.d survival script"
81    blockdev --setrw /dev/block/mapper/system$SLOT 2>/dev/null
82    mount -o rw,remount /system
83    ADDOND=/system/addon.d/99-magisk.sh
84    cp -af $COMMONDIR/addon.d.sh $ADDOND
85    chmod 755 $ADDOND
86  fi
87
```

Figura 23: Magisk criando um arquivo que permite identificá-lo.
Fonte: Magisk.[37]

Conhecendo esse arquivo pode-se realizar uma melhor implementação da detecção de arquivos (Seção 2.4.2), e assim, é possível identificar muitos aparelhos *rooteados* com o Magisk.

Contudo, como esse diretório (/system/addon.d) não existe em todos os Androids e, além disso, outros arquivos podem aparecer de acordo com cada sistema, é interessante utilizar a metodologia apresentada para criar uma lista de arquivos a serem verificados.

Esse procedimento de busca por nome e conteúdo de arquivos, foi repetido em algumas outras ROMs instaladas no aparelho de teste (da Seção 4.1.2), e em uma delas foi possível encontrar outro arquivo que identifica o Magisk, o "/system/media/theme/miui_mod_icons/com.topjohnwu.magisk.png", porém esse é criado pelo próprio Android modificado e permanece mesmo que o Magisk Manager tenha seus nomes alterados (como demonstrado na Figura 11).

A Tabela 4 resume as ROMs que foram analisadas e os arquivos encontrados nelas que podem ser utilizados para identificar o Magisk.

[37] Disponível em <https://github.com/topjohnwu/Magisk/blob/3e4caabecb324f6375a031505e5b8bbf5b859aa6/scripts/flash_script.sh#L83>. Acessado em 6 de outubro de 2020.

ROM	Versão	Arquivos
MIUI	11.0.6.0	(nenhum arquivo identificado)
Pixel Experience	10.0-20200531-14-31	/system/addon.d/99-magisk.sh
Resurrection Remix	8.5.9-20200822	/system/addon.d/99-magisk.sh
Xiaomi EU	12.0.3.0	/system/media/theme/miui_mod_icons/com.topjohnwu.magisk.png

Tabela 4: Arquivos que indicam a presença do Magisk.
Fonte: O autor.

5.2.2 Checagem de Ícones

Checagem de ícones é uma forma de verificação proposta por este trabalho que não foi encontrada na revisão da literatura. Ela pode ser classificada como um subtipo da checagem abordada na Seção 2.4.1, visto que o objetivo de ambas é verificar se um determinado aplicativo (que tem relação com o *root*) está instalado no aparelho.

O diferencial desta checagem é que os aplicativos instalados no aparelho são analisados através de seus ícones. Dessa forma, o ícone de cada um deles é comparado com o do Magisk Manager.

Existem vários algoritmos que podem ser utilizados para comparar imagens, um deles é o pHash [33], que produz um hash para cada imagem analisada, e posteriormente, permite que diferentes hashes tenham suas distâncias calculadas, com o objetivo de informar o grau de semelhança entre as imagens.

A fim de realizar uma prova de conceito, foi desenvolvido um aplicativo que implementa essa checagem e está atualmente disponível no GitHub.[38] A Figura 24 demonstra uma das funções que foram utilizadas nele, a *checkIcons*, que é responsável por obter os ícones dos aplicativos instalados, enviá-los para a função *getMagiskIconDistance* e então decidir, pelo valor de distância obtido, se o aplicativo analisado é o Magisk Manager.

[38] Disponível em <https://github.com/dr01dr4v3/magisk-icon-checker>. Acessado em 7 de outubro de 2020.

```
private boolean checkIcons() {
  PackageManager pm = getPackageManager();
  List<ApplicationInfo> packages = pm.getInstalledApplications(0);
  /* para cada aplicativo instalado */
  for (ApplicationInfo info : packages) {
      try {
          String pkgName = info.packageName;
          Drawable icon = pm.getApplicationIcon(pkgName);
          int distance = getMagiskIconDistance(icon);
          addText("[*] d: " + distance + ", p: " + pkgName);
          /* quanto menor a distancia,
             mais semelhantes sao as imagens */
          if (distance <= 1) {
              addText("\n\n[+] Magisk Manager encontrado!\n" +
                      "-> " + pkgName + "\n\n");
              return true;
          }
      } catch (Exception e) {  }
  }
  addText("\n[-] Magisk Manager não encontrado!\n");
  return false;
}
```

Figura 24: Buscando o acesso *root* através de ícones.
Fonte: O autor.

A Figura 25 ilustra a função *getMagiskIconDistance*, que utilizando o algoritmo pHash, calcula o hash de um ícone específico e então o avalia contra alguns modelos de ícones do Magisk.

```
private int getMagiskIconDistance(Drawable icon) {
  int shorterDistance = Long.SIZE;
  long hashApp = phash.getHash(icon);
  /* comparando icone recebido com os do magisk */
  for (long magiskHash : magiskHashes) {
      int curDistance = phash.distance(hashApp, magiskHash);
      /* salvando a menor distancia, que equivale
         ao maior grau de semelhanca encontrado */
      shorterDistance = Math.min(shorterDistance, curDistance);
  }
  return shorterDistance;
}
```

Figura 25: Obtendo a maior semelhança de um ícone arbitrário com o do Magisk.
Fonte: O autor.

Apesar de o Magisk Manager ter apenas um ícone, é possível que o mesmo sofra alterações por temas que são instalados no aparelho. Portanto, a função

getMagiskIconDistance também utilizava outros ícones que são comumente utilizados por temas, eles são ilustrados na Figura 26.

Figura 26: Ícones utilizados na função *getMagiskIconDistance.*
Fonte: O autor.

Então, o aplicativo implementado para a prova de conceito foi executado em diversos aparelhos (incluindo ambos do Capítulo 4) e em todos teve sucesso em identificar o Magisk Manager quando ele realmente existia. A Tabela 5 mostra os ambientes em que ele foi testado.

Aparelho	**ROM**	**Configuração**	**Resultado do Aplicativo**
Redmi K20	Pixel Experience (10.0-20200531-14-31)	Magisk (20.4) com o aplicativo prova de conceito na lista do MagiskHide	Root detectado
Redmi K20	Xiaomi EU (12.0.3.0) com tema de ícones	Magisk (20.4)	Root detectado
Samsung Galaxy Note 5	Resurrection Remix (6.2.1-20181007)	Magisk (21.0)	Root detectado
Redmi Note 4	Resurrection Remix (6.2.1-20181107)	Magisk (20.4) com Magisk Manager com *package name* aleatório	Root detectado
Moto G4 Plus	Resurrection Remix (6.2.0-20180916)	Aparelho *rooteado* sem o Magisk	Root não detectado

Tabela 5: Ambientes em que o aplicativo prova de conceito foi executado.
Fonte: O autor.

Percebe-se que a técnica de checagem de ícones só funciona quando o aparelho *rooteado* recebe um aplicativo com um ícone, o que acontece quando o *root* é obtido através do Magisk. Porém, para dispositivos *rooteados* de outra forma,

como ocorreu com o Moto G4 Plus (da Seção 4.2.2), as técnicas tradicionais são necessárias e, portanto, é recomendado que elas sejam utilizadas junto com essa.

Finalmente, a Figura 27 apresenta a única tela do aplicativo desenvolvido como prova de conceito, que mostra as distâncias calculadas para cada ícone analisado no aparelho e o resultado no fim do processo. Pode-se observar que o Magisk Manager foi encontrado mesmo estando com um *package name* aleatório.

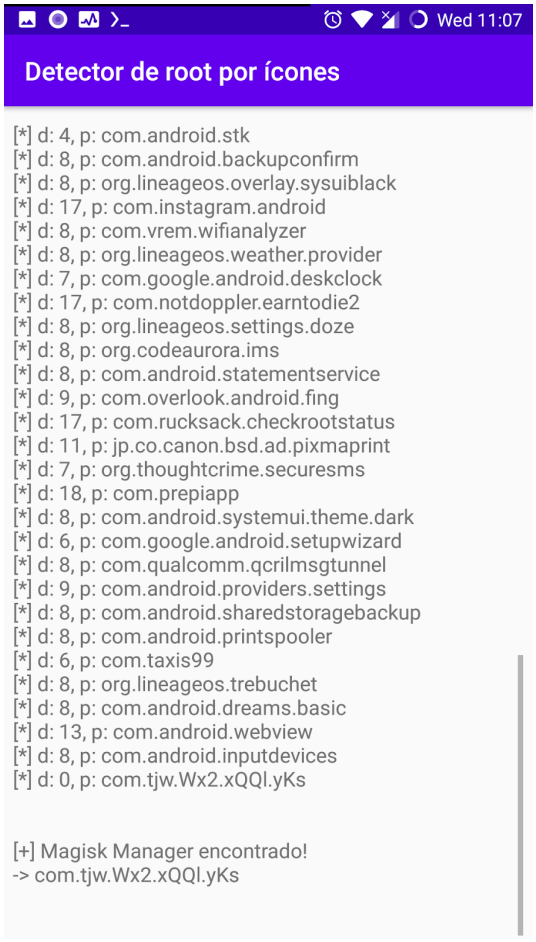


Figura 27: Aplicativo prova de conceito.
Fonte: O autor.

6. Conclusões e Trabalhos Futuros

Existem muitas pessoas que habilitam o acesso *root* em seus aparelhos Android. Contudo, esse é um procedimento que gera consequências negativas do ponto de vista de segurança. Portanto, os desenvolvedores Android começaram a utilizar técnicas para detectar o *root*, e posteriormente, surgiram métodos para realizar a evasão dessas técnicas.

O objetivo geral deste trabalho foi investigar o estado da arte e da prática das técnicas para detecção de *root*, avaliar a eficácia delas contra os métodos de evasão e propor melhorias para o lado da detecção. Esse objetivo foi alcançado com êxito, visto que a revisão literária indicou as principais técnicas utilizadas, o *survey* e dois estudos empíricos realizados demonstraram que atualmente a detecção está em desvantagem nessa disputa e, por fim, sugestões de melhorias para a detecção foram apresentadas.

O primeiro objetivo específico foi revisar a literatura em busca de técnicas de evasão e detecção de *root*. Foram encontradas 6 principais técnicas, além de um serviço do Google, para a detecção e o Magisk e algumas outras ferramentas para a evasão.

O segundo objetivo específico foi conduzir um *survey* para identificar técnicas utilizadas por desenvolvedores, na detecção, e por usuários, na evasão. Os resultados do *survey* indicaram que a detecção não está tão madura quanto a evasão, e também, que o Magisk é a ferramenta mais usada para esconder o *root*.

O terceiro objetivo específico foi analisar aplicativos, a fim de avaliar a atual eficácia da detecção contra métodos de evasão identificados. Ao ser alcançado, através da realização de dois estudos empíricos, foi reforçada a ideia de que a maturidade da detecção ainda é consideravelmente inferior à da evasão visto que apenas um aplicativo, dentre centenas analisados, conseguiu detectar o *root*.

O quarto e último objetivo específico foi encontrar características de aparelhos *rooteados* que poderiam ser utilizadas para melhorar a detecção. Esse objetivo foi atingido ao se encontrar duas particularidades que podem ser usadas para detectar o Magisk, que são um arquivo criado por ele e o ícone de seu aplicativo.

Foi possível observar que a evasão está em vantagem na disputa contra a detecção, com o Magisk conseguindo esconder o *root* contra quase todos aplicativos. Adicionalmente as implementações das técnicas de detecção ainda parecem pouco maduras, visto que além de não serem eficazes, a preocupação com elas ocorre em grande apenas parte no nicho de aplicativos financeiros. Finalmente, foram sugeridas duas abordagens para melhorar a detecção, que se mostraram eficazes contra o Magisk.

Uma das limitações deste trabalho ocorreu no *survey*, que não teve um número tão expressivo de desenvolvedores respondendo as perguntas não

obrigatórias, e assim, pode ter alcançado um resultado menos preciso da compreensão da detecção.

Outra limitação aconteceu na análise dos aplicativos mais baixados, onde aqueles que não apresentaram um indicativo visual da detecção ou os que o faziam apenas após uma autenticação, devem ter sido classificados de forma incorreta.

Em relação a trabalhos futuros, pode ser interessante realizar entrevistas com desenvolvedores que já trabalharam detectando *root*, em busca de extrair mais informações sobre suas experiências.

Outra investigação importante é repetir o estudo empírico realizado com os aplicativos mais baixados, porém agora tentando utilizar credenciais e meios para verificar detecções de *root* silenciosas.

Por fim, um trabalho futuro valioso é o de procurar heurísticas que permitam à busca por nome e conteúdo de arquivos ser feita de forma automatizada, sem a avaliação direta de um humano sobre falsos-positivos.

7. Referências

[1] Statista. **Smartphone users worldwide 2020**. Disponível em < https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/ >. Acessado em 29 de março de 2020.

[2] Statista. **Mobile OS market share 2019**. Disponível em < https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/ >. Acessado em 29 de março de 2020.

[3] Statista. **Global Android malware volume 2018**. Disponível em < https://www.statista.com/statistics/680705/global-android-malware-volume/ >. Acessado em 29 de março de 2020.

[4] Kaspersky Lab. **Rooting your Android: Advantages, disadvantages, and snags**. Disponível em < https://www.kaspersky.com/blog/android-root-faq/17135/ >. Acessado em 29 de março de 2020.

[5] CASATI, Luca; VISCONTI, Andrea. **The Dangers of Rooting: Data Leakage Detection in Android Applications**. Mobile Information Systems, 2018.

[6] EVANS, Nathan; BENAMEUR, Azzedine; SHEN, Yun. **All your Root Checks are Belong to Us: The Sad State of Root Detection**. Proceedings of the 13th ACM International Symposium on Mobility Management and Wireless Access, p. 81-88. 2015.

[8] SensePost. **objection**. Disponível em < https://github.com/sensepost/objection >. Acessado em 29 de março de 2020.

[9] Kaspersky Lab. **Classificações de malware**. Disponível em < https://www.kaspersky.com.br/resource-center/threats/malware-classifications >. Acessado em 2 de abril de 2020.

[10] WRIGHT, Arol. **Pokémon GO's aggressive root checking now looks for the existence of a TWRP and Titanium Backup folder**. Disponível em < https://www.xda-developers.com/pokemon-go-aggressive-root-checking-twrp-folder/ >. Acessado em 2 de abril de 2020.

[11] Google. **Android Debug Bridge (adb)**. Disponível em < https://developer.android.com/studio/command-line/adb?hl=pt-br >. Acessado em 23 de setembro de 2020.

[13] KOTIPALLI, Srinivasa Rao; IMRAN, Mohammed. **Hacking Android**. Packt Publishing, 2016.

[15] Termux. **Termux root packages**. Disponível em < https://github.com/termux/termux-root-packages/tree/master/packages >. Acessado em 16 de maio de 2020.

[16] KASUNIC, Mark. **Designing an Effective Survey**. Carnegie Mellon University, 2005.

[17] XDA Developers. **How to Root Any Device**. Disponível em < https://www.xda-developers.com/root/ >. Acessado em 23 de maio de 2020.

[18] WU, John. **Magisk**. Disponível em < https://github.com/topjohnwu/Magisk >. Acessado em 23 de maio de 2020.

[19] NGUYEN-VU, Long; CHAU, Ngoc-Tu; KANG, Seongeun; JUNG, Souhwan. **Android rooting: An arms race between evasion and detection**. Security and Communication Networks, 2017.

[20] SUN, San-Tsai; CUADROS, Andrea; BEZNOSOV, Konstantin. **Android rooting: Methods, detection, and evasion**. Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices, p. 3-14. 2015.

[21] EVANS, Nathan; BENAMEUR, Azzedine; SHEN, Yun. **All your root checks are belong to us: The sad state of root detection**. Proceedings of the 13th ACM International Symposium on Mobility Management and Wireless Access, p. 81-88. 2015.

[22] ELENKOV, Nikolay. **Android Security Internals An In-Depth Guide to Android's Security Architecture**. No Starch Press, 2015.

[23] LINFO. **Kernel Definition**. Disponível em < http://www.linfo.org/kernel.html >. Acessado em 07 de setembro de 2020.

[27] Google. **SafetyNet Attestation API**. Disponível em < https://developer.android.com/training/safetynet/attestation >. Acessado em 27 de setembro de 2020.

[28] SIMÕES, Pedro. **SuperSU foi eliminado da Play Store e é o fim da melhor app de root do Android**. Disponível em <https://pplware.sapo.pt/smartphones-tablets/android/supersu-google-play-root-android/>. Acessado em 28 de setembro de 2020.

[29] Linux man page. **ptrace**. Disponível em < https://linux.die.net/man/2/ptrace >. Acessado em 29 de setembro de 2020.

[30] WikiMili. **Linux namespaces**. Disponível em < https://wikimili.com/en/Linux_namespaces >. Acessado em 29 de setembro de 2020.

[31] Frida. **Frida**. Disponível em < https://frida.re/ >. Acessado em 30 de setembro de 2020.

[32] Guardsquare. **DexGuard vs ProGuard**. Disponível em < https://www.guardsquare.com/en/blog/dexguard-vs-proguard >. Acessado em 7 de outubro de 2020.

[33] The Hacker Factor Blog. **Looks Like It**. Disponível em < https://www.hackerfactor.com/blog/index.php?/archives/432-Looks-Like-It.html >. Acessado em 7 de outubro de 2020.

APÊNDICE A - SURVEY

Bloco 1: Perfil do participante
Nessa seção são coletadas informações sobre sua experiência com desenvolvimento Android.

1) Você trabalha ou já trabalhou no desenvolvimento de aplicativos Android?

- ○ Sim, e também já trabalhei com detecção de root no Android.
- ○ Sim, mas nunca tive experiência com detecção de root.
- ○ Não.

2) Qual o seu tempo de experiência com desenvolvimento Android?

- ○ Nunca trabalhei com desenvolvimento android.
- ○ Menos de 6 meses.
- ○ Entre 6 meses e 1 ano.
- ○ Entre 1 e 2 anos.
- ○ Entre 2 e 3 anos.
- ○ Entre 3 e 5 anos.
- ○ Mais de 5 anos.
- ○ Outros.

Bloco 2: Mecanismos utilizados para detectar acesso root
Nessa seção são coletadas informações sobre sua experiência com técnicas para detectar o acesso root.

3) Você utilizou alguma biblioteca para a detecção de root?

- ☐ Não.
- ☐ Sim, RootTools (Stericson).
- ☐ Sim, RootBeer (scottyab).
- ☐ Sim, jail-monkey (GantMan).
- ☐ Sim, meat-grinder (DimaKoz).
- ☐ Outros.

4) Se você já implementou a detecção por conta própria, quais as principais verificações que eram realizadas? (Ex: busca por arquivo X, busca por Y na lista de aplicativos instalados)

5) Quando você adiciona a detecção em um aplicativo, qual costuma ser o contexto que gera essa necessidade?

6) Na sua opinião, qual é a melhor forma para um aplicativo realizar a detecção de acesso root e por quê?

7) Você acredita que as formas de detecção que você conhece poderiam ser burladas ou talvez gerar falsos-positivos? De que forma?

8) Você gostaria de fornecer algum comentário adicional?

Bloco 2: Acesso root
Nessa seção são coletadas informações sobre sua experiência em esconder o acesso root.

3) Você já utilizou alguma técnica para esconder, para um aplicativo, o fato de que o dispositivo (ex: celular, tablet) tinha acesso root ?

- ○ Sim.
- ○ Não, nunca tive dispositivos rooteados.
- ○ Não, apesar de que já tive/tenho dispositivos rooteados.

Bloco 3: Mecanismos utilizados para esconder acesso root
Nessa seção são coletadas informações sobre sua experiência com técnicas para esconder o acesso root.

4) Que mecanismo você já usou para esconder o acesso root? (ex: aplicativo, framework ou ferramentas envolvidas)

- ☐ Magisk.
- ☐ objection (sensepost).
- ☐ fridantiroot.
- ☐ Outros.

5) Se possível, informe aplicativos contra os quais você conseguiu esconder o root, e se já houve algum para o qual você não conseguiu.

6) Você gostaria de fornecer algum comentário adicional?

APÊNDICE B - APLICATIVOS DE CHECAGEM DE ROOT ANALISADOS

#	Aplicativo	Package Name	Versão
1	Advanced Root Checker	com.anu.developers3k.rootchecker	1.4.0
2	Android Root Check	checkroot.andreasch.com.androidcheckroot	1.1
3	APK Extractor	com.adria.apkextractor	1.3.5
4	App Info	com.buildchecker.rootchecker.appinfo	4.0
5	Basic Root Checker	com.iboalali.basicrootchecker	v1.5vc20
6	Check For Root	com.mavenmaverick.checkforroot	1.0
7	Check Root	mk.dimitar.checkroot	5.2.0
8	Check Root - Network	arcturus.studio.checkroot.network	2.25.34
9	Check Root Status	com.lotej.rootstatus	1.0
10	Check Root Status for Android	com.rucksack.checkrootstatus	0.2.4.15
11	Device Info	com.droidbest.deviceinfo	1.0
12	Device Info 360	com.vishtekstudios.deviceinfo	3.0.1
13	Dev Root	devcodenet.in.devroot	1.0
14	Easy Root Check	com.protpsyche.easyrootcheck	1.0
15	Easy ROOT CHECKER	com.Rooteasy.chkerpro	7.0
16	Got Root?	com.urmilshroff.rootchecker	2.0
17	IS Rooted	esso.App.isRoot	1.0
18	King Checker	com.theking.forroot.checkerappjdiim	9.5.1
19	King Go Root	com.kinggoroot.checker	4.1.0
20	KRoot Checker	com.shajazteam.rootverifier	5.6
21	KRoot Checker Tool	hasdev.com.krootcheckertool	1.3
22	Magisk checker Root	com.checker.man	1.0
23	MAGISK DETECTOR	com.pat.detector	1.3
24	MAGISK MANAGER CHECKER	com.checkermag.iskmanager	2.3
25	Native Root Checker	com.kozhevin.rootchecks	1.0.1
26	RootbeerFresh	com.kimchangyoun.rootbeerFresh.sample	0.10
27	RootBeer Sample	com.scottyab.rootbeer.sample	0.8
28	Root Check	com.paraspoit.rootchecker	1.0.0
29	Root Check	com.avrapps.rootcheck	1.0
30	Root Check	fi.kask.rootcheck	1.1
31	Root Check	com.amosmobile.rootcheck	1.4

32	Root Check	com.jrummyapps.rootchecker	4.4.1.0
33	root_checker	com.appybuilder.sandipkhade453.root_checker	1.0.0
34	Root Checker	com.kyumpany.rootchecker	1.0.0
35	Root Checker	com.zedstudioapps.RootCheckerBusyBoxChecker	1.0.0
36	Root Checker	com.askfortricks.rootcheck	1.0.13
37	RootChecker	com.andreacioccarelli.rootchecker	1.0.1
38	Root Checker	com.rvbanna420.Root_Checker	1.01
39	RootChecker	com.capture4me.rootchecker	1.0.2
40	Root Checker	com.flinkapp.root.checker	1.0.2
41	Root checker	fast.root.checker	1.0.2
42	Root Checker	com.appfactorykim.rootchecker	1.0.3
43	Root Checker	acs.pt.rootchecker	1.0.5
44	Root Checker	com.acr.rootchecker	1.0.5
45	Root Checker	com.appybuilder.kumarkeshavk1999.RootChecker	1.0
46	Root Checker	com.device.root.checker	1.0
47	Root Checker	com.innovguru.rootcheck	1.0
48	Root Checker	com.json200.rootchecker	1.0
49	Root Checker	com.libertines.rootchecker	1.0
50	Root Checker	com.mobile.rootchekers	1.0
51	Root Checker	com.mpdevelopers.rootchecker	1.0
52	Root Checker	com.mta.cpuinfo.root.checker	1.0
53	Root Checker	net.bytefreaks.rootchecker	1.0
54	Root Checker	rootstatus.bahdev.com.rootchecker	1.0
55	Root Checker	simpleapps.rootchecker	1.0
56	Root Checker	com.apps.easy.rootchecker	1.1
57	Root Checker	com.atominvention.rootchecker	1.1
58	Root Checker	com.loganbassett.rootchecker	1.1
59	Root Checker	com.rootmydevice.com	1.1
60	Root Checker	com.theappmachine.rootcheck	1.1
61	Root Checker	com.tiagorlampert.rootchecker	1.1
62	Root Checker	com.yy.adam.rootchecker	1.1
63	Root checker	ztronn.com.rootchecker	1.1
64	Root Checker	com.arunpaul.rootchecker	1.2
65	Root Checker	com.zayed.ltd.rootchecker	13.0
66	Root Checker	com.shxosj.rootchecker	1.3

67	Root Checker	com.barmybytes.rootchecker	1.4
68	Root Checker	com.busybox.rootchecker	1.4
69	Root Checker	softwareexpertise_data.RootChecker	1.5
70	Root Checker	appsmachine.rootcheckerandroid	1.6
71	Root Checker	com.afe.rootchecker	1.6
72	Root Checker	com.fa.RootChecker	1.6
73	Root Checker	com.lineports.Root_Checker	17.1.0
74	Root Checker	com.apps.quarex.rootbusyboxchecker	1.8
75	Root Checker	checker.root.com.myapplication	2.0
76	Root Checker	com.clivin.rootchecker	2.0
77	Root Checker	com.qbysoft.rootchecker	2.0
78	Root Checker	com.root1.com	2.0
79	Root Checker	com.phongphan.rootchecker	2.2
80	Root Checker	com.root.rootcheck	2.9.6
81	Root Checker	com.easyapps.earootchecker	30.1217
82	Root CHecKeR	com.funstuff.rootchkr	3.0
83	Root Checker	org.freeandroidtools.root_checker	3.3.1
84	RootChecker	com.icondice.rootchecker	3.6
85	Root Checker	com.intelligent.rootandroid	3
86	Root Checker	com.bitbyte.rootchecker	4.0
87	Root Checker	com.vvlabsrootchecker.appy	4.2.0
88	Root Checker	burrows.apps.rootchecker	4.3.0
89	Root Checker	com.rodolphestore.rootchecker	8.0.0
90	Root Checker	com.rootchecker.superz	8.0
91	Root Checker	com.cmdann.rootcheckerpro	8.1
92	Root Checker - Advanced Root Checker	com.clubofcinema.rootcheckeradvancedrootchecker	1.0
93	Root Checker Basic	com.exploudapps.rootchecker	1.0
94	Root Checker Basic	com.joeykrim.rootcheck	6.4.8
95	Root Checker for Android	com.pathak.rootchecker	1.0
96	Root Checker Free	com.thunkable.android.tboyslusher.Root_Checker	1.0
97	Root Checker Free	com.didar.mobile.rootcheckerfree	1.1
98	Root Checker New Free	com.rootche.ckermanager	2.1
99	Root checker Pro	com.minhquang.rootcheckerpro	1.0
100	Root Checker Pro	com.root.checker.advanced.pro	1.0
101	Root Checker - Pro	com.techdreamers.rootchecker	1.1

102	Root Checker Pro	com.sbacham.srinu.rootcheckerpro	1.6
103	Root Checker Pro	com.mango.rootvalidator	2.0.32
104	Root Checker- super su	com.ss.root.checker	1.0.5
105	Root Check Fast SU Checker	com.bit.rootcheckfastsuchecker.android.gp	2.0
106	Root Check & Info	com.haiderart.rootchecker	3.0
107	Root Detector	com.gh05tcom.rootdector	1.1
108	Root Detector	shin2.rootdetector	1.1
109	Root Essentials	com.superthomaslab.rootessentials	2.4.9
110	Root Prüfer (DE)	de.owsianowski.rootcheck	2.2.0
111	Rootseed	com.ahyad.root	2.0
112	Root Status	com.androidxda.rootstatus	4.0.0
113	Root Status Check	com.abk.rootcheckerpro	1.11
114	Rootsu	com.nextappsgen.rootsu	2.0.1
115	Root Tester	it.davidev.roottester	1.31M
116	Root Toolbox	com.dreamdroid.droidroot	1.0.2
117	Root Validator	com.mobiappsstudios.validator	1.0.1
118	Root Validator	eu.thedarken.rootvalidator	3.1
119	Root Verifier	com.loxeras.rootverifier	1.0.1
120	Root Verifier	com.abcdjdj.rootverifier	2.0
121	Root Verify	br.com.pogsoftwares.rootverify	3.02
122	SafetyNet And Verify Apps Validation	com.rucksack.safetynetandverifyappsvaldiation	0.2.4.11
123	SafetyNet `attest`	com.scottyab.safetynet.sample	0.7
124	SafetyNet Checker	com.flinkapps.safteynet	1.0.8
125	SafetyNet Test	org.freeandroidtools.safetynettest	1.2.1
126	Simple Root Check	com.thirstydevs.simplerootcheck	1.0
127	Simple Root Check	rirozizo.simplerootcheck	1.8
128	Simple Root Checker	com.graffixnyc.rootchecker	1.8.1
129	SK Root Checker	com.shark.skrootcheckerfree	1.0
130	SUBinary	hpandro.java.infosec.su	1.0.1
131	Super ROOT Checker	com.superroot.checker	4.0
132	Super Simple Root Check	ssrc.eyespyblock.com	2
133	SU Root Checker	com.pricefull.surootchecker	2.0
134	Verify Root	com.fluffydelusions.app.verifyroot	1.0.2
135	Verify root	com.loggerapps.Verify_root	1.0.4
136	Verify Root State	com.tokozea.cekstatusroot	1.2

APÊNDICE C - APLICATIVOS MAIS BAIXADOS QUE FORAM ANALISADOS

Categoria	Aplicativos
Art and Design	com.canva.editor, jp.ne.ibis.ibispaintx.app, com.sec.penup, com.appsilicious.wallpapers, com.vblast.flipaclip, com.phone.launcher.lite, com.medibang.android.paint.tablet, com.eyewind.paperone, com.sweetsugar.pencileffectfree, com.jdpapps.textt1, com.brakefield.painter, pl.planmieszkania.android, com.fashion.tattoo.name.my.photo.editor, com.adobe.spark.post, com.tattoo.maker.design.app, com.creative.Learn.to.draw.flowers, com.redberry.glitterdressa2, com.phone.launcher.android, air.com.KalromSystems.SandDrawLite, com.sweefitstudios.drawgraffiti
Auto and Vehicles	com.google.android.projection.gearhead, com.aa.extreme.city.gt.car.stunts, com.autoscout24, ru.gibdd_pay.app, kz.kolesa, com.taxsee.taxsee, de.mobile.android.app, com.devplank.masterplaca, com.lelic.speedcam, com.offroad.jeep.driving.simulator, com.city.coach.bus.simulator.drive, com.stormbreakxr.snow.mountain.bike.racing.motocross.race, ru.farpost.dromfilter, org.reactivephone, jabi.pdd2, com.girnarsoft.cardekho, com.carwale, com.gigigo.ipirangaconectcar, br.com.icarros.androidapp
Beauty	com.arcsoft.perfect365, com.perfectcorp.ycn, com.northpark.beautycamera, com.lyrebirdstudio.beauty, com.Fashion.Nails.Girls.Game, kr.co.company.hwahae, com.piupiuapps.hairstyles, com.barilab.handmirror.googlemarket, jp.hotpepper.android.beauty.hair, com.luckystars.hairstylesstepbystep, com.jkfantasy.camera.jkpmirrorcamera, google.free.icequeen.peachygamesllc, com.dsrtech.macho, best.photo.app.weddinghairstyles2018, com.dsrtech.menhairstyles, com.trove, br.art.code.meucronogramacapilar, pcm.light.crown.photo.editor
Books and Reference	com.google.android.apps.books, wp.wattpad, com.sirma.mobile.bible.android, com.audible.application, com.amazon.kindle, org.wikipedia, com.nhn.android.search, com.google.android.stardroid, com.mobisystems.msdict.embedded.wireless.oxford.dictionaryofenglish, org.jw.jwlibrary.mobile, br.biblia, com.andi.alquran.id, com.dictionary, com.bestweatherfor.bibleoffline_pt_ra, com.quran.labs.androidquran, kjv.bible.kingjamesbible, HinKhoj.Dictionary, com.scribd.app.reader0, com.merriamwebster, tv.telepathic.hooked
Business	com.linkedin.android, com.facebook.pages.app, com.mobisystems.office, com.indeed.android.jobsearch, com.mobisystems.fileman, com.google.android.apps.meetings, us.zoom.videomeetings, com.ubercab.driver, com.microsoft.teams, com.netqin.ps, com.ticno.olymptrade, com.asus.ia.asusapp, com.dataviz.docstogo, com.metropcs.service.vvm, naukriApp.appModules.login, com.cisco.webex.meetings, com.adobe.scan.android, ru.hh.android, net.slideshare.mobile
Comics	com.naver.linewebtoon, com.zalivka.animation2, com.nhn.android.webtoon, com.kauf.talking.baum.TalkingJamesSquirrel, com.iconology.comics, net.daum.android.webtoon, com.appmag.endless.temple.princessfinaljungleozrunfrozen, br.com.escolhatecnologia.voztradutor, com.webcomics.manga, com.energysh.drawshow, com.rookiestudio.perfectviewer, com.marvel.comics, com.lezhin.comics, com.colorjoy.learn.to.draw.glow.comics, com.sundev.memestickers, learn.to.draw.glow.cartoon

Communication	com.whatsapp, com.android.chrome, com.google.android.gm, com.facebook.orca, com.skype.raider, com.google.android.apps.tachyon, com.google.android.talk, com.google.android.apps.messaging, com.sec.android.app.sbrowser, com.UCMobile.intl, com.viber.voip, com.truecaller, jp.naver.line.android, com.imo.android.imoim, org.telegram.messenger, com.facebook.mlite
Dating	com.zoosk.zoosk, com.okcupid.okcupid, com.jaumo.prime, com.hotornot.app, dating.app.chat.flirt.wgbcv, com.hitwe.android, com.jaumo.casual, ru.mobstudio.andgalaxy, com.hily.app, ru.fotostrana.sweetmeet, com.matchlatam.parperfeitoapp, com.choiceoflove.dating, com.dating.android, com.once.android, com.ipart.android, de.appfiction.yocutiegoogle, mingle.android.mingle2, com.planetromeo.android.app
Education	com.duolingo, com.microblink.photomath, com.google.android.apps.classroom, com.byjus.thelearningapp, com.youdao.hindict, com.memrise.android.memrisecompanion, com.CultureAlley.japanese.english, co.brainly, com.babbel.mobile.android.en, com.ruangguru.livestudents, com.brainbow.peak.app, com.unacademyapp, example.matharithmetics, com.busuu.android.enc, com.joytunes.simplypiano, com.fusionprojects.edmodo, com.wonder, com.quizlet.quizletandroid
Entertainment	com.google.android.play.games, com.ismaker.android.simsimi, tv.twitch.android.app, com.bitstrips.imoji, com.digidust.elokence.akinator.freemium, com.outfit7.talkingangelafree, com.outfit7.talkinggingerfree, com.amazon.avod.thirdpartyclient, com.outfit7.talkingben, com.baviux.voicechanger, com.outfit7.talkingnewsfree, com.google.android.apps.youtube.kids, com.graymatrix.did
Events	com.greetingsisland.sam, com.gametime.gametime, com.vividseats.android, com.kpuripemilu2019, com.todaytix.TodayTix, com.sympla.tickets, com.topRingtones.TikTok, draziw.leftday.widget, countdown.reminder.widget, com.letsdogether.dogether, splendid.invitation.maker, com.conceptworks.spontacts, com.tickpickllc.ceobrien.tickpick, com.chic.colorfuldiscolights2, com.sony.psexp2016, com.ingresse.ticketbooth, sk.ipndata.meninyamenafree
Finance	net.one97.paytm, ru.sberbankmobile, com.phonepe.app, com.paypal.android.p2pmobile, br.com.bb.android, br.gov.caixa.tem, com.bradesco, br.com.gabba.Caixa, com.mobikwik_new, com.iqoption, com.sbi.lotusintouch, br.gov.caixa.auxilio, com.csam.icici.bank.imobile, com.sbi.SBIFreedomPlus, com.chase.sig.android, com.itau, com.santander.app
Food and Drink	com.application.zomato, com.ubercab.eats, br.com.brainweb.ifood, in.swiggy.android, com.yelp.android, com.mcdonalds.mobileapp, com.dominospizza, com.grability.rappi, com.dd.doordash, com.nzn.tdg, com.starbucks.mobilecard, com.joelapenna.foursquared, com.mcdo.mcdonalds, com.sei.android
Game Action	com.dts.freefireth, com.imangi.templerun2, com.tencent.ig, com.mobile.legends, com.fungames.sniper3d, com.nekki.shadowfight, com.supercell.brawlstars, com.activision.callofduty.shooter, air.com.hypah.io.slither, com.yodo1.crossyroad, com.miniclip.bowmasters, com.h8games.helixjump, com.fdgentertainment.bananakong, com.outfit7.talkingtomgoldrun, com.miniclip.agar.io, com.gameloft.android.ANMP.GloftM5HM, com.kabam.marvelbattle, com.playgendary.kickthebuddy
Game Adventure	com.roblox.client, com.nianticlabs.pokemongo, com.prettysimple.criminalcaseandroid, com.explorationbase.ExplorationLite, com.gameloft.android.ANMP.GloftIVHM, com.playmini.miniworld, com.fingersoft.benjibananas, com.tellurionmobile.realmcraft, free.os.jump.superbros.adventure.world, com.igoldtech.streetchaser, com.superbinogo.jungleboyadventure, com.tinyco.potter, com.frogmind.badland, com.naturalmotion.clumsyninja, com.telltalegames.walkingdead100, com.glu.stardomkim, com.g5e.hiddencity.android, com.tinyco.familyguy, com.rtsoft.growtopia

Game Arcade	com.kiloo.subwaysurf, com.imangi.templerun, com.fgol.HungrySharkEvolution, com.robtopx.geometryjumplite, net.mobigame.zombietsunami, com.halfbrick.fruitninjafree, com.sega.sonicdash, com.halfbrick.jetpackjoyride, com.mediocre.smashhit, com.nordcurrent.canteenhd, com.nekki.vector, com.dvloper.granny, com.FDGEntertainment.redball4.gp, com.natenai.glowhockey, com.alien.shooter.galaxy.attack, io.voodoo.paper2, com.mojang.minecrafttrialpe, io.voodoo.crowdcity
Game Board	com.ludo.king, com.jetstartgames.chess, com.pixel.art.coloring.color.number, com.superking.parchisi.star, uk.co.aifactory.chessfree, paint.by.number.pixel.art.coloring.drawing.puzzle, com.moonfrog.ludo.club, pl.lukok.draughts, com.LoopGames.Domino, com.dimcoms.checkers, com.tictactoe.wintrino, com.mobirix.SnakeGame, com.blacklightsw.ludo, net.peakgames.Yuzbir, net.peakgames.OkeyPlus, com.fungamesforfree.colorfy, air.com.buffalo_studios.newflashbingo, air.com.bitrhymes.bingo, com.ahoygames.okey
Game Card	com.mobilityware.solitaire, com.playtika.wsop.gp, com.matteljv.uno, com.kathleenOswald.solitaireGooglePlay, io.teslatech.callbreak, jp.konami.duellinks, com.blizzard.wtcg.hearthstone, com.tfgco.games.strategy.free.castlecrush, com.dragonplay.liveholdempro, com.king.pyramidsolitairesaga, com.gsn.android.tripeaks, com.mobilityware.spider, com.igg.bzbee.deckheroes, com.blyts.trucolite.activities, air.ru.crazypanda.wpcm, com.konami.ygodgtest, com.spacegame.solitaire.basic, com.lemongame.klondike.solitaire, com.rstgames.durak
Game Casino	com.zynga.livepoker, air.com.playtika.slotomania, com.octro.teenpatti, com.leftover.CoinDozer, com.teenpatti.hd.gold, com.pacificinteractive.HouseOfFun, com.murka.scatterslots, com.luckyday.app, com.williamsinteractive.jackpotparty, com.playstudios.myvegas, com.huuuge.casino.slots, com.doubleugames.DoubleUCasino, com.droidhen.game.poker, com.playtika.caesarscasino, com.genina.android.blackjack.view, com.neptune.domino, com.dragonplay.slotcity, com.dummy.poker, com.selfawaregames.acecasino, com.igg.pokerdeluxe
Game Casual	com.king.candycrushsaga, com.outfit7.mytalkingtomfree, com.outfit7.mytalkingangelafree, me.pou.app, com.supercell.hayday, com.gameloft.android.ANMP.GloftDMHM, com.playrix.gardenscapes, com.king.farmheroessaga, com.playrix.homescapes, com.playrix.township, com.king.candycrushsodasaga, com.ea.game.pvz2_row, com.rovio.baba, com.ea.game.simpsons4_row, com.moonactive.coinmaster, com.king.petrescuesaga, com.midasplayer.apps.bubblewitchsaga2, com.outfit7.mytalkingtom2, com.outfit7.talkingtom, com.outfit7.mytalkinghank
Game Educational	com.tocaboca.tocakitchen2, jp.co.ofcr.cm00, com.sinyee.babybus.shopping, com.edujoy.masha.games, air.com.peppapig.paintbox, com.sinyee.babybus.tailor, com.gokids.transportbuilding, com.razmobi.monstertrucks2, com.color.mandala, com.tocaboca.tocalifeworld, mytown.home, com.libiitech.princesssalon, com.pepiplay.pepihome, com.PepiPlay.PepiShopping, com.sinyee.babybus.care, com.sinyee.babybus.restaurant, com.PepiPlay.KingsCastle, com.GamesForKids.Mathgames.MultiplicationTables, com.sinyee.babybus.drinks
Game Music	com.youmusic.magictiles, com.amanotes.beathopper, com.gismart.realpianofree, com.amanotes.pamadancingroad, com.grillgames.guitarrockhero, com.eyu.piano, com.gismart.guitar, com.orange.kidspiano.music.songs, com.bigbluebubble.singingmonsters.full, com.ubisoft.dance.JustDance, air.com.freshplanet.games.WaM, com.wordsmobile.musichero, com.nojoke.realpianoteacher, com.joyjourney.PianoWhiteGo, com.gismart.realdrum2free, com.studio7775.BeatMP3, com.amanotes.db, com.Zeeppo.GuitarBand, com.pennypop.dance, com.studio7775.BeatMP3v2

Game Puzzle	com.playrix.fishdomdd.gplay, com.mind.quiz.brain.out, com.zeptolab.ctr.ads, com.zeptolab.ctr2.f2p.google, com.disney.wheresmywater2_goo, com.bitmango.rolltheballunrollme, com.king.candycrushjellysaga, com.bigduckgames.flow, com.differencetenderwhite.skirt, com.water.balls, com.shootbubble.bubbledexlue, com.game5mobile.lineandwater, com.supertapx.lovedots, com.app.rescuecut, net.peakgames.toonblast, net.peakgames.amy, com.king.bubblewitch3, com.europosit.pixelcoloring, com.disney.frozensaga_goo, com.matchington.mansion
Game Racing	com.fingersoft.hillclimb, com.gameloft.android.ANMP.GloftA8HM, com.skgames.trafficrider, com.ea.games.r3_row, com.skgames.trafficracer, com.ansangha.drdriving, com.ea.game.nfs14_row, com.rovio.angrybirdsgo, com.fingersoft.hcr2, com.aim.racing, com.topfreegames.bikeracefreeworld, com.creativemobile.DragRacing, com.vectorunit.purple.googleplay, com.julian.fastracing, com.slippy.linerusher, com.combineinc.streetracing.driftthreeD, com.wordmobiles.bikeRacing, com.droidhen.game.racingmoto, com.ansangha.drparking4, com.wordsmobile.RealBikeRacing
Game Role Playing	com.fungames.flightpilot, com.netmarble.mherosgb, com.nekki.shadowfight3, com.lockwoodpublishing.avakinlife, com.com2us.smon.normal.freefull.google.kr.android.common, com.cocoplay.sweet.bakery, com.solou.catendless.run, com.YovoGames.dentist, com.lilithgame.hgame.gp, com.makingfun.mageandminions, com.ea.game.starwarscapital_row, com.cjenm.monster, com.nexon.da3.global, com.my.hc.rpg.kingdom.simulator, com.gamehivecorp.taptitans, air.MSPMobile, com.moonton.magicrush, com.plarium.raidlegends, com.cjenm.sknights
Game Simulation	es.socialpoint.DragonCity, com.ea.games.simsfreeplay_row, com.fungames.blockcraft, com.mgc.miami.crime.simulator.two, com.ea.game.simcitymobile_row, com.miniclip.plagueinc, com.episodeinteractive.android.catalog, com.fluffyfairygames.idleminertycoon, com.gameloft.android.ANMP.GloftDOHM, com.ludia.jurassicworld, com.ovilex.bussimulator2015, com.ea.gp.simsmobile, com.ovilex.eurotruckdriver, com.clean.road, com.giantssoftware.fs14, com.secretexit.turbodismount, com.i6.FlightSimulatorAirplane3D, com.zuuks.bus.simulator.ultimate, com.gameloft.android.ANMP.GloftDYHM, com.parking.game
Game Sports	com.miniclip.eightballpool, com.ea.gp.fifamobile, com.firsttouchgames.story, com.miniclip.carrom, com.billiards.city.pool.nation.club, com.touchtao.soccerkinggoogle, com.miniclip.footballstrike, com.forthblue.pool, com.threed.bowling, com.mgc.runnergame, com.junerking.archery, jp.konami.pesam, eu.nordeus.topeleven.android, com.ea.game.easportsufc_row, com.nextwave.wcc2, com.ea.gp.nbamobile, com.miniclip.soccerstars, com.dnddream.headsoccer.android, com.mobirix.fishinghook
Game Strategy	com.supercell.clashofclans, com.supercell.clashroyale, com.igg.android.lordsmobile, com.ea.game.pvzfree_row, com.igg.castleclash, com.maxgames.stickwarlegacy, com.supercell.boombeach, net.wargaming.wot.blitz, es.socialpoint.MonsterLegends, com.hcg.cok.gp, com.machinezone.gow, com.socialquantum.acityint, air.com.goodgamestudios.empirefourkingdoms, com.aa.generaladaptiveapps, com.epicwaronline.ms, com.longtech.lastwars.gp, com.yottagames.mafiawar, com.quarterborrowresult.customer, com.empire.grow.rome, br.com.tapps.bidwars
Game Trivia	com.etermax.preguntados.lite, com.unicostudio.braintest, logos.quiz.companies.game, com.scimob.ninetyfour.percent, com.quizup.core, com.orangenose.trick, com.xmonetize.quizzland, lemmingsatwork.quiz, se.feomedia.quizkampen.de.lite, com.ninetyfour.degrees.app, com.freepuzzlegames.logoguessing.quiz, com.gartic.Gartic, com.etermax.kingdoms, com.works.timeglass.logoquiz, com.etermax.trivia.preguntados2, fr.two4tea.fightlist, com.guilardi.eusei, guess.the.brand.logo.quiz.icomania

Game Word	com.zynga.words, com.fugo.wow, de.lotum.whatsinthefoto.es, de.lotum.whatsinthefoto.us, com.omgpop.dstfree, com.fanatee.cody, com.quelaba.sopaletras, com.word.puzzle.game.connect, com.word.game.fun.puzzle.prison.escape.captain, com.melimots.WordSearch, com.bitmango.go.wordcookies, com.zytoona.wordscrush, se.maginteractive.wordbrain, com.wordgame.words.connect, com.peoplefun.wordcross, com.andron.crosswords2, com.zynga.wwf2.free, com.zytoona.lostword2, com.hbwares.wordfeud.free, com.wordloco.wordchallenge
Health and Fitness	com.sec.android.app.shealth, com.popularapp.periodcalendar, nic.goi.aarogyasetu, homeworkout.homeworkouts.noequipment, com.huawei.health, com.myfitnesspal.android, org.iggymedia.periodtracker, com.xiaomi.hm.health, com.runtastic.android, sixpack.sixpackabs.absworkout, com.fitbit.FitbitMobile, loseweight.weightloss.workout.fitness, com.google.android.apps.fitness, com.nike.plusgps, com.clue.android, com.endomondo.android, cc.pacer.androidapp, com.strava, com.northpark.drinkwater, com.popularapp.thirtydayfitnesschallenge
House and Home	codematics.universal.tv.remote.control, com.houzz.app, com.ivuu, com.move.realtor, com.tekoia.sure.activities, com.urbanclap.urbanclap, com.trulia.android, es.roid.and.trovit, com.alexvas.dvr, com.chbreeze.jikbang4a, kr.co.station3.dabang, fr.anuman.HomeDesign3D, com.idealista.android, com.anuntis.fotocasa, ru.cian.main, com.autodesk.homestyler, com.trulia.android.rentals, net.bucketplace, com.warden.cam
Libraries and Demo	com.google.samples.apps.cardboarddemo, dz.condor.Condorpassport, inc.trilokia.pubgfxtool.free, galaxys9ringtones.ringtonesfors9.notifications, com.studio.coolmaster.coolerapp.cooling, com.famousringtones2017.populartones.bestringtones, com.quanticapps.quranandroid
Lifestyle	com.pinterest, com.tinder, com.google.android.apps.chromecast.app, com.samsung.android.oneconnect, com.bitsmedia.android.muslimpro, com.ftw_and_co.happn, com.life360.android.safetymapd, com.adpog.diary, Com.sktelecom.minit, com.masarat.salati, org.reyfasoft.reinavalera1960, com.AppRocks.now.prayer, com.mobilexsoft.ezanvakti, com.dramaton.slime
Maps and Navigation	com.ubercab, com.waze, com.grabtaxi.passenger, com.olacabs.customer, com.sygic.aura, com.taxis99, com.tranzmate, ru.yandex.taxi, ru.yandex.yandexmaps, sinet.startup.inDriver, com.careem.acma, com.grabtaxi.driver2, com.mapfactor.navigator, com.rapido.passenger, br.com.easytaxi, com.here.app.maps, com.cuvora.carinfo, com.ubercab.uberlite, ee.mtakso.client
Medical	com.lbrc.PeriodCalendar, app.bpjs.mobile, tr.com.innova.fta.mhrs, com.webmd.android, com.alodokter.android, com.ada.app, com.linkdokter.halodoc.android, com.NetmedsMarketplace.Netmeds, ru.mobiledimension.kbr, com.smsrobot.period, com.practo.fabric, com.AnatomyLearning.Anatomy3DViewer3, com.ladytimer.ovulationcalendar, com.lybrate.phoenix, com.medscape.android
Music and Audio	com.google.android.music, com.spotify.music, com.soundcloud.android, com.smule.singandroid, com.shazam.android, com.gaana, com.jio.media.jiobeats, deezer.android.app, com.studiosol.palcomp3, com.springwalk.mediaconverter, tunein.player, com.amazon.mp3, com.dywx.larkplayer, com.google.android.apps.youtube.music, com.starmakerinteractive.starmaker, com.melodis.midomiMusicIdentifier.freemium
News and Magazines	com.twitter.android, com.google.android.apps.magazines, flipboard.app, com.eterno, com.newsdog, in.AajTak.headlines, com.til.timesnews, com.sony.nfx.app.sfrc, com.zinio.mobile.android.reader, com.reddit.frontpage, com.toi.reader.activities, com.quora.android, com.waveline.nabd, id.co.babe

Parenting	com.babycenter.pregnancytracker, com.hp.pregnancy.lite, org.findmykids.app, com.oubapps.po.ch, com.becloser, com.zoemob.gpstracking, com.fsp.android.h, com.healofy, com.dokdoapps.mybabypiano, com.tickledmedia.ParentTown, us.mitene, au.com.penguinapps.android.babyfeeding.client.android, com.liveyap.timehut, com.realdream.kidspolice_sp, com.urbandroid.babysleep, com.wachanga.babycare, com.amila.parenting, com.wachanga.pregnancy, com.screentime.rc
Personalization	net.zedge.android, com.gau.go.launcherex, com.apusapps.launcher, com.lbe.parallel.intl, home.solo.launcher.free, com.jb.gokeyboard, com.ogqcorp.bgh, com.yandex.browser, com.google.android.apps.wallpaper,com.teslacoilsw.launcher,com.xinmei365.font, com.simejikeyboard, com.herman.ringtone, com.transsion.XOSLauncher, com.wallpaperscraft.wallpaper
Photography	com.google.android.apps.photos, com.picsart.studio, com.linecorp.b612.android, com.camerasideas.instashot, com.venticake.retrica, vStudio.Android.Camera360, com.commsource.beautyplus, com.cyberlink.youcammakeup, com.joeware.android.gpulumera, io.faceapp, com.cyworld.camera, vsin.t16_funny_photo, com.cyberlink.youperfect, com.zentertain.photoeditor, com.cam001.selfie, com.lyrebirdstudio.montagenscolagem, jp.naver.linecamera.android, com.adobe.psmobile, com.instagram.boomerang
Productivity	com.google.android.apps.docs, com.microsoft.office.word, com.microsoft.office.excel, com.microsoft.skydrive, com.microsoft.office.powerpoint, com.google.android.calendar, com.sec.app.samsungprintservice, com.adobe.reader, com.touchtype.swiftkey, com.dropbox.android, com.google.android.apps.docs.editors.docs, com.google.android.keep, com.google.android.apps.docs.editors.sheets, com.microsoft.office.onenote, com.google.android.apps.docs.editors.slides
Shopping	com.flipkart.android, com.contextlogic.wish, com.alibaba.aliexpresshd, com.mercadolibre, com.lazada.android, com.ebay.mobile, com.myntra.android, com.joom, com.snapdeal.main, club.fromfactory, com.amazon.mShop.android.shopping, com.zzkko, com.alibaba.intl.android.apps.poseidon, com.google.zxing.client.android, com.allgoritm.youla, com.avito.android
Social	com.facebook.katana, com.instagram.android, com.snapchat.android, com.zhiliaoapp.musically, com.facebook.lite, com.google.android.apps.plus, com.vkontakte.android, sg.bigo.live, com.tumblr, ru.ok.android, in.mohalla.sharechat, com.ss.android.ugc.boom, com.pof.android, com.myyearbook.m, com.askfm
Sports	com.cricbuzz.android, com.espn.score_center, com.gotv.nflgamecenter.us.lite, de.motain.iliga, com.scores365, com.mobilefootie.wc2010, com.sofascore.results, com.livescore, com.fifa.fifaapp.android, com.kokteyl.goal, com.resultadosfutbol.mobile, com.activaweb.matchendirect, com.kokteyl.mackolik, es.lfp.gi.main, com.july.cricinfo, com.fivemobile.thescore, com.pl.premierleague, com.dazn, com.digiturk.ligtv, com.uefa.ucl
Tools	com.google.android.gms, com.google.android.googlequicksearchbox, com.google.android.marvin.talkback, com.google.android.tts, com.lenovo.anyshare.gps, com.google.android.inputmethod.latin, com.google.android.apps.translate, com.google.android.apps.nbu.files, com.google.android.deskclock, com.google.android.calculator
Travel and Local	com.google.android.apps.maps, com.google.android.street, com.booking, com.google.earth, com.tripadvisor.tripadvisor, com.gojek.app, com.comuto, com.ixigo.train.ixitrain, com.mapswithme.maps.pro, ru.yandex.yandexnavi, com.airbnb.android, ru.dublgis.dgismobile, mobi.wifi.toolbox, com.makemytrip, com.goibibo, com.trivago

Video Players	com.google.android.youtube, com.google.android.videos, com.mxtech.videoplayer.ad, video.like, com.quvideo.xiaoying, com.xvideostudio.videoeditor, com.utorrent.client, com.nexstreaming.app.kinemasterfree, com.mobilemotion.dubsmash, org.videolan.vlc, com.stupeflix.replay, com.kwai.video, com.zhiliaoapp.musically.go, all.video.downloader.allvideodownloader, free.video.downloader.freevideodownloader
Weather	com.weather.Weather, com.devexpert.weather, com.accuweather.android, de.wetteronline.wetterapp, com.droid27.transparentclockweather, com.handmark.expressweather, com.yahoo.mobile.client.android.weather, com.macropinch.swan, com.aws.android, com.chanel.weather.forecast.accu, com.apalon.weatherlive.free, com.ilmeteo.android.ilmeteo, com.graph.weather.forecast.channel, ru.yandex.weatherplugin, aplicacion.tiempo, com.exovoid.weather.app