Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

DroidXP: Um benchmark para apoiar a pesquisa em mineração de sandboxes Android

Marcos Vinicius P. Marques

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientador
Prof. Dr. Rodrigo Bonifácio de Almeida

Brasília
2023

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

DroidXP: Um benchmark para apoiar a pesquisa em mineração de sandboxes Android

Marcos Vinicius P. Marques

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof. Dr. Rodrigo Bonifácio de Almeida (Orientador)
CIC/UnB

Me. Handrick da Costa
CIC/UnB

Me. Pedro Costa
CIC/UnB

Prof. Dr. Marcelo Grandi Mandelli
Coordenador do Bacharelado em Ciência da Computação

Brasília, 01 de março de 2023

Dedicatória

Dedico esta conquista à minha mãe, Noeme, primeiramente, que me proporcionou todo o apoio necessário durante minha jornada acadêmica para que eu, Marcos, chegasse até aqui. Em segundo lugar, dedico ao meu orientador, Rodrigo, pelo suporte e pela atenção que prestou no desenvolvimento deste trabalho. E por último, mas não menos importante, dedico ao meu mentor Tiago Stutz, por compartilhar sua experiência e me ajudar a tomar decisões na minha carreira.

Agradecimentos

Primeiramente, gostaria de agradecer à Universidade de Brasília, onde tive a oportunidade de estudar e desenvolver meu projeto. Agradeço aos professores que me orientaram e compartilharam seus conhecimentos e experiências, bem como aos colegas de turma que contribuíram para minha formação acadêmica.

Também gostaria de agradecer aos meus familiares e amigos, que me apoiaram e encorajaram durante todo o processo de elaboração da dissertação. Seus conselhos e palavras de incentivo foram fundamentais para minha motivação e perseverança.

Não poderia deixar de agradecer aos profissionais e pesquisadores que contribuíram com sua experiência e conhecimento na área estudada. Suas contribuições foram essenciais para o desenvolvimento do meu trabalho.

A todos que contribuíram para a elaboração deste trabalho, expresso minha profunda gratidão e reconhecimento. Muito obrigado!

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), por meio do Acesso ao Portal de Periódicos.

Resumo

O uso universal dos smartphones elevou, na mesma proporção, a quantidade de usuários do Android, um sistema operacional projetado para dispositivos eletrônicos móveis que, devido à sua popularidade e acessibilidade, tornou-se pouco seguro para quem o utiliza. Ou seja, com a popularização do sistema, inúmeras falhas de segurança passaram a ser identificada, tais como casos de malware. Esse movimento incentivou pesquisadores, especialmente da área de segurança da computação, a uma jornada intensa de busca por ferramentas capazes de combater essas ocorrências que colocam em risco dados sigilosos como, por exemplo, dados pessoais e bancários de seus usuários. Foi pensando nisso que se criou a ideia de experimentar o DroidXP, um software que permite, aos estudiosos da área, comparar simplificadamente ferramentas de geração para teste de sandboxes. Dessa forma, o experimento apresentado neste trabalho permitiu revelar que o Sapienz supera outras ferramentas de geração de casos de teste.

Palavras-chave: Android, Benchmark, Segurança, DroidXP, Sistema, Usuários

Abstract

The universal use of smartphones has increased, in the same proportion, the number of Android users, an operating system designed for mobile electronic devices that, due to its popularity and accessibility, has become unsafe for those who use it. That is, with the popularization of the system, numerous security flaws began to be identified, such as cases of malware. This movement encouraged researchers, especially in the area of computer security, to undertake an intense search for tools capable of combating these occurrences that put confidential data at risk, such as, for example, personal and banking data of its users. It was with this in mind that the idea of trying out DroidXP was created, a software that allows experts in the field to easily compare generation tools for testing sandboxes. Thus, the experiment presented in this work revealed that Sapienz outperforms other test case generation tools.

Keywords: Android, Benchmark, Security, DroidXP, System, User.

Sumário

Lista de Figuras

Capítulo 1

Introdução

Android é um sistema operacional que foi criado no ano de 2005, principalmente, para atender dispositivos móveis, como tablets e smartphones. Esse nome se deu por alusão a "Androides sonham com ovelhas elétricas?", de Philip K. Dick, um romance futurista com androides de inteligência semelhante à de seres humanos – nome que foi pensado em uma passagem de seus primeiros fundadores pela Google. Contudo, apesar de ter sido gerado em 2005, desde 2003, a história do sistema operativo já vinha sendo construída, quando houve a criação da empresa Android Inc, por Rich Miner, Nick Sears, Chris White e Andy Rubin [1].

Em 2008, foi lançado e disponibilizado, nas lojas, o primeiro aparelho portando um sistema operacional Android e poucos anos depois o Android se tornou referência, competindo diretamente com a IOS, sistema da Apple – líder, em 2022, ocupando 1º lugar no ranking de 100 marcas mais valiosas do mundo todo. A primeira versão do Android foi a 1.0, totalmente diferente das versões que conhecemos e utilizamos atualmente. Porém, ainda assim, já se destacava com recursos bem avançados para a época, como por exemplo a função de copiar texto e colar, de baixar aplicativos pela loja da Google e de acessar as plataformas bem populares Youtube e outros players de música (exemplo: Spotify) [2] [3] [4].

A popularização do sistema operacional Android foi tão acelerada, que em três anos, 50% da população já utilizava aparelhos Android e hoje [1] [3], é a plataforma móvel mais amplamente utilizada, assumindo a liderança no mercado de smartphones, tablets e outros dispositivos eletrônicos, principalmente devido à sua interface simplificada e quanto à programação de aplicativos compatíveis com o sistema entre a comunidade de desenvolvedores.

A partir dessa aceleração, especialmente entre os anos de 2011 e 2014, diversas falhas de segurança causando ataques de malware foram descobertas por pesquisadores da área e algumas empresas que comercializavam antimalwares, o que transformou o tema "segu-

rança" em um campo altamente atraente e pertinente aos estudiosos da época. Entende-se por malware, uma ação – ou um software malicioso – que tem, como objetivo, causar danos ao dispositivo [4, 5]. É o que hackers – especialistas e sistema de informação – fazem, por exemplo, quando possuem intenção de invadir um sistema para captar os dados pessoais ou bancários de alguém por meio das movimentações que esse usuário faz utilizando seu smartphone. Consequentemente, antimalwares são softwares que protegem contra esses ataques maliciosos [6].

Atualmente, o Android ocupa liderança em uso de sistema operacional para smartphones em relação a seus concorrentes – iOs (principalmente, sistema operacional da Apple e concorrente direto do Android) e Blackberry, por exemplo. São 75% de usuários e consumidores Android.

Ao contrário da loja de aplicativos do mercado da Apple, o Google Play – loja digital oficial com mais de um milhão de opções de aplicativos compatíveis com Android – não verifica os aplicativos carregados manualmente. Ao contrário disso, o Google Play conta com o Bouncer, um ambiente de emulação dinâmico para se proteger contra ameaças de aplicativos mal-intencionados. Ele forneceria proteção contra ameaças, mas não pode analisar a vulnerabilidade dos aplicativos existentes. Aplicativos maliciosos podem enganar aplicativos vulneráveis para divulgar informações privadas do usuário que prejudicam inadvertidamente a reputação deste último.

Os ataques maliciosos têm, desde então, utilizado maneiras avançadas de contornar a funcionalidade dos mecanismos de segurança já disponíveis para sistema Android. Por isso é fundamental compreender os padrões de uso recentes e utilizar esses estudos como um ponto de partida deste experimento [7].

A partir da compreensão de como esses softwares funcionam, com o intuito de encontrar meios de erradicar esses efeitos maliciosos e danosos da popularização de uso do Android, foi que surgiu a necessidade de desenvolver um benchmarking, para investigar a eficácia de sandboxes de mineração, usando ferramentas de teste automatizadas, a fim de comparar o desempenho de ferramentas de geração de teste com sandboxes de mineração.

Neste trabalho, serão apresentados esses experimentos de benchmarking, tendo como solução uma ferramenta desenvolvida para assegurar os dados que são salvos em aplicativos como bancos, redes sociais, WhatsApp, e outros compatíveis com Android e que exijam a segurança de dados. Ao longo do desenvolvimento deste estudo, será analisada a estrutura da ferramenta e suas funcionalidades, bem como os resultados obtidos por meio dos testes realizados na prática do uso, além de compará-la com outras concorrentes ou que apresentem ideais semelhantes no mercado.

Antes de dar início à pratica dos experimentos, vale conceituar o que é segurança da informação, sua importância e quais pilares essa proteção defende. O termo "segurança da

informação" significa o conjunto de práticas que protegem dados sensíveis que deveriam ser mantidos em sigilo – incluindo dados pessoais, que são respaldados pela Lei Geral de Proteção de Dados (LGPD, Lei nº 13.709/2018, lei cujo objetivo é proteger os direitos fundamentais de liberdade e privacidade), por exemplo.

Existem cinco pilares na segurança da informação. São eles:

1. Confidencialidade: proteção de dados cujos acessos não foram autorizados. Por meio de autenticação de senha, verificação biométrica e processos de criptografia, este pilar controla o acesso.

2. Integridade: proteção da integridade dos dados, preservando a originalidade das informações. Ou seja, assegura que as informações não sejam alteradas sem devida autorização.

3. Disponibilidade: a fim de proteger os dados contra quaisquer imprevistos como blecautes, por exemplo, este pilar zela pela estabilidade de acesso permanentemente aos dados do sistema.

4. Autenticidade: pilar que protege a legitimidade dos responsáveis pelas informações. Ou seja, garante a autoria verdadeira dos dados. Irretratabilidade: pilar que defende a "não negação" de autoria.

5. Responsabilidade: assegura a responsabilidade por ações vinculadas aos dados. Em seguida, foi preciso compreender a história do sistema operacional Android, como surgiu e sua popularização, aprofundamento que será realizado mais à frente.

A partir desse ponto, é que a problematização do tema será construída a fim de que as análises sejam intencionais e efetivas como soluções ou, pelo menos, início de uma jornada resolutiva para o problema levantado. É o que será exposto a seguir, na próxima etapa deste trabalho.

1.1 Contextualização

Em 2022, foi publicada uma pesquisa – pela empresa Cibersegurança Check Point – com os dez agentes maliciosos mais comuns em infectar smartphones no Brasil. Dentre eles, com 42,19% e 6,88%, respectivamente, foram identificados os malwares Emotet e Chaes, responsáveis por captar informações bancárias do usuário. Isso quer dizer que os dois principais agentes infecciosos de Android, mesmo quando se considera pesquisas recentes a respeito do tema, ainda estão associados a roubo de dados bancários [7].

O sistema operacional Android, objeto de pesquisa deste estudo, gerou um alto ganho monetário para empresários do mercado devido à sua popularização, o que despertou hackers quanto aos "ganhos" que eles poderiam ter por meio dos ataques de malware.

Foi publicado um alerta, em agosto de 2022, pela Polícia Federal, a respeito de golpes capazes de esvaziar a conta bancária vinculada ao celular – brecha encontrada por hackers quando o usuário instala um programa malicioso em seu smartphone, uma prática cada vez mais comum considerando a quantidade de anúncios que um indivíduo recebe em seu aparelho diariamente, incentivando essas ações de instalação. Confirma-se isso quando a superintendência da Polícia Federal de Pernambuco informa, junto ao alerta publicado, que uma média de 40 mil brasileiros foram vítimas desse golpe [7].

Movimentações como essas envolvem a atuação da Polícia Federal, pois se trata de crimes cibernéticos, que se trata exatamente de práticas conduzidas por hackers mal-intencionados, com aplicação de técnicas avançadas de tecnologia que visam a prática de um crime, como por exemplo, a invasão da privacidade de alguém ou o próprio roubo de dados.

Além das recomendações divulgadas pela Polícia Federal, em agosto de 2022, há outras práticas que podem reduzir as chances de se tornar vítima de malwares. São elas: a utilização de "senhas fortes" para acessar contas de e-mails, bancos ou redes sociais, por exemplo, com letras maiúsculas somadas a números e ícones; utilização de recursos como "Face ID", que se trata do reconhecimento facial para acessar a conta e atualização frequente do software [8]. Contudo, a partir do alto índice de crimes cibernéticos que acontecem todos os dias no país, é possível observar que mesmo com a popularização do sistema Android e as recomendações popularmente divulgadas, quiçá por conta disso, ainda existe uma fragilidade muito presente quanto à segurança do software, e o mais importante, que coloca em risco a vida pessoal e financeira de seus usuários, que têm buscado praticidade na rotina, vinculando todas as contas em seus respectivos aparelhos por meio de aplicativos.

Há diversas maneiras pelas quais um malware pode ser inserido em um aplicativo compatível com Android. Uma dessas maneiras é por meio da modificação do aplicativo legítimo, adicionando código malicioso à sua estrutura. Esse aplicativo modificado é então distribuído em fontes não oficiais, como sites de terceiros ou redes peer-to-peer, onde os usuários o baixam acreditando que se trata de uma versão legítima do aplicativo[8].

É importante lembrar que a Google Play Store – ambiente virtual de busca e download de aplicativos compatíveis com sistema operacional Android – é um ambiente seguro para baixar aplicativos, pois os aplicativos são verificados antes de serem disponibilizados para download. Em 2022, por exemplo, o Google anunciou que impediu 1 milhão e 200 mil aplicativos, em média, de serem violados. No entanto, nem todos os malwares são

detectados durante essa verificação, então mesmo se tratando de um ambiente seguro, é possível que aplicativos maliciosos sejam encontrados na loja de aplicativos da Google ou até mesmo que sejam aplicativos seguros, contudo, vítimas desses ataques. Então, é importante que os usuários tomem precauções ao baixar aplicativos, verificando a reputação do desenvolvedor e lendo as avaliações dos usuários antes do procedimento, porém infelizmente esses cuidados não garantem 100% da segurança dos seus dados.

O objetivo mais recorrente por ataques cibernéticos em aplicativos Android é exatamente o roubo de dados. Afinal, com os dados pessoais e bancários de um usuário, é possível realizar diversos outros furtos e outras invasões maliciosas. De acordo com o Relatório de Ameaças da ESET Internet Security (empresa de proteção última geração) que saiu em 2021, os ataques com foco em roubo de dados bancários aumentaram em 428% com malwares dos tipos [7]:

1. Trojans bancários: mecanismo capaz de burlar sistemas de autenticação com o intuito de roubar dados bancários e enviar esses dados ao(s) servidor(es) do(s) usuário(s) invasor(es).

2. Android Ransomware: ataque malicioso que geralmente invade o dispositivo e criptografa arquivos importantes. Muito utilizado para extorquir o usuário vítima, ou seja, geralmente o invasor pede uma recompensa em pagamento para permitir que o usuário recupere os arquivos perdidos.

3. RATs (Remota Access Tojans): ataque que visa bisbilhotar as movimentações do usuário, gravando a tela, por exemplo, para captar diversas informações como senhas, credenciais, dados confidenciais, etc.

Entre 2010 e 2013, houve um considerável aumento no índice de penetrações de malware em sistemas Android, o que consequentemente, despertou de maneira ainda mais emergente a necessidade de se estudar e propor soluções de segurança e mitigação mais robustos e eficientes para aplicativos compatíveis com Android. Afinal, um ataque malware só acontece quando há vulnerabilidade de segurança no sistema.

Diante disso, a indústria antimalware também passou a ter grande contribuição na exploração de vulnerabilidades em busca de combate aos ataques, por meio de métodos de assinatura que tendem a contornar os códigos com novas assinaturas para cada amostra maliciosa, razão pela qual mecanismos contra esses ataques precisam atualizar frequentemente a base de dados de assinaturas.

Em 2011, William Enck estudou algumas soluções de segurança, em especial, a possibilidade da permissão para que houvesse quaisquer comunicações entre aplicativos. Apesar de não ter conseguido propor mecanismos totalmente eficientes, William complementou

suas pesquisas e análises com orientações futuras a quem se interessasse a dar continuidade ao trabalho[8].

Com base nessa contextualização a respeito da segurança dos aplicativos compatíveis com sistema Android e suas vulnerabilidades, foi que surgiu a ideia de criar o benchmarking proposto neste trabalho.

1.2 Origem do Problema

Diagnosticado o principal problema enfrentado pela popularização do sistema Android, diversas técnicas passaram a ser desenvolvidas com o intuito de identificar esses softwares maliciosos e combater as vulnerabilidades enfrentadas pela plataforma móvel. Por exemplo, alguns pesquisadores exploraram o uso de ferramentas de análises dinâmicas e geradores de caso de teste para minerar sandboxes – ambiente virtual seguro que tende a ser utilizado para a realização de experimentos, com o objetivo de proteger usuários do Android desses comportamentos maliciosos.

A abordagem Mining Android Sandbox (aqui chamada de abordagem MAS) aproveita ferramentas de geração automatizada de casos de teste para explorar o comportamento de um aplicativo em termos de chamadas a APIs sensíveis e, em seguida, gera uma sandbox Android. Neste artigo, usaremos os termos Aplicações Android, Android Apps e Apps de forma intercambiável para se referir a aplicações de software Android. Durante a execução normal do aplicativo, a sandbox pode bloquear qualquer chamada a uma API sensível que não tenha sido observada durante a fase exploratória. Os pesquisadores mostraram que a abordagem MAS também é eficaz na detecção de uma classe popular de malware Android que "repacota"aplicativos benignos - isto é, começando com uma versão benigna de um aplicativo de uma loja de aplicativos oficial, como o Google Play, pode-se infectá-lo com código malicioso, como transmitir informações sensíveis para um servidor privado. O aplicativo infectado pode então ser compartilhado com usuários usando diferentes lojas de aplicativos. Trabalhos de pesquisa anteriores comparam a precisão de sandboxes Android para detecção de malware produzidas a partir da execução de diferentes ferramentas de geração de casos de teste, incluindo as ferramentas Monkey, DroidBot e Droidmate. Por meio desse ciclo, quanto mais eficiente for a ferramenta de geração de teste (por exemplo, em termos de cobertura de código), mais precisa a sandbox tende a ser em identificar uma ação de malware. Esses relatórios trazem evidências de que a ferramenta de geração de testes DroidBot supera as outras ferramentas, levando a sandboxes que classificam como malware 70% dos aplicativos repacotados em um conjunto de dados.

Ou seja, junto dos avanços de pesquisa na área de geração de casos de teste, novas soluções vêm constantemente surgindo, profissionais (grupo onde nos inserimos) passaram

a encontrar dificuldades técnicas para reproduzir estudos anteriores que comparam as ferramentas de geração de casos de teste com as de mineração de sandboxes. Com isso, entende-se que este problema ocorre, principalmente, devido à falta de um suporte de referência, que assumiria a função de auxiliar esses pesquisadores ao longo da tarefa oportuna.

Para cada usuário, o sistema Android identifica a aplicação com um código único, o que significa que quaisquer criações que não estejam no SDCard (Secure Digital Card) do dispositivo só pode ser acessado pela própria criação. Com isso, um ataque malicioso consegue acessar com facilidade um SQLite (base de dados com código aberto) sem criptografia, para enviar os dados a um servidor web, por exemplo, especialmente quando se usa as mesmas senhas e logins para aplicativos diversos.

A partir desse diagnóstico, o próximo passo é colocar em prática, então, os experimentos, analisar os resultados e discutir a eficácia da funcionalidade da ferramenta que foi desenvolvida.

Capítulo 2

Objetivos

2.1 Objetivo Geral

Estes experimentos têm, como objetivo principal, contribuir ativamente com pesquisadores que buscam encontrar meios de reduzir ou erradicar o alto índice de falhas de segurança enfrentados pelo Android por meio do uso de aplicativos compatíveis com o sistema. Afinal, o crescimento acelerado de usuários do sistema foi o que gerou a necessidade de mais segurança, e a partir disso, a necessidade dos experimentos.

Para alcançar o objetivo esperado, serão realizados dois estudos envolvendo pesquisas, experiências e avaliação dos resultados obtidos, que serão compartilhados ao longo deste trabalho de curso sinalizados por etapas de background, benchmarking, instrumentação, execução, análise e conclusão do estudo.

Na primeira etapa, chamada de background, será observado como a ferramenta se comporta. Ou seja, é o momento de detalhar quais são as características dessa ferramenta, sua composição, compatibilidade e como ela influencia no funcionamento do aplicativo. Em seguida, parte-se para a etapa do benchmarking – como o já induzido pelo próprio nome – que diz respeito ao processo de análise da ferramenta, não só olhando para seu desenvolvimento, mas principalmente em comparação a outras, com uma análise mais profunda do que já deu certo ou não com o uso de outras ferramentas semelhantes. Além disso, é também na etapa do benchmarking que testes de aperfeiçoamento são aplicados, após ter se observado o comportamento da ferramenta na etapa do background.

As etapas seguintes são divididas entre a instrumentação e a execução, que se tratam da parte "mão na massa" do projeto. Até então, por mais que existam alguns testes, são mais baseados em estudar e observar o desempenho da ferramenta. Já nessas duas etapas, há execução de alguns procedimentos de análise estática e repetição do experimento para cada geração de caso de teste, ferramenta e arquivos, com o intuito de corrigir falhas identificadas anteriormente e otimizar as funcionalidades da ferramenta estudada.

O trabalho será encerrado com a análise e a conclusão do estudo, onde será exposto se as expectativas a respeito da ferramenta foram atendidas em sua totalidade ou em parte, quais resultados foi possível alcançar e se algum recurso foi insuficiente.

Por fim, caso não seja possível colocar em prática todos os experimentos, abrir-se-á uma seção que chamaremos de "trabalhos futuros", com o intuito de orientar e nortear profissionais interessados da área que busquem estes estudos como um ponto de partida, mas por outro lado, tenham o objetivo em dar continuidade, aperfeiçoando as funcionalidades ou testando ferramentas compatíveis que possam atuar em parceria com a de benchmarking, por exemplo.

2.2 Objetivos Específicos

A partir do diagnosticado já existente desde a popularização do Android, em meados de 2011, criou-se a necessidade de estudos mais direcionados ao que tange à segurança de dispositivos portadores de Android. Como uma abordagem focada em mitigar esse problema, estes experimentos apresentam o "Droidxp", um benchmark – estudo de práticas – criado para apoiar pesquisadores e profissionais na integração de ferramentas de geração de casos de teste e na comparação de seu desempenho em sandboxes de mineração com a plataforma Android.

Dentre os objetivos desta criação, destaca-se o de auxiliar pesquisadores e profissionais na reprodução de estudos de pesquisa neste campo. Em linhas gerais, a principal contribuição do Droidxp é facilitar a reprodução dos estudos empíricos, permitindo que os pesquisadores integrem as ferramentas de geração de casos de teste ratificadas.

Um outro objetivo a ser alcançado por meio deste experimento é garantir que usuários amadores – que não sejam estudiosos na área de sistemas de segurança – possam utilizar dispositivos portadores de Android sem que se tornem reféns de malwares, ou seja, dos danos maliciosos que podem ser causados por um software não confiável.

Ademais, pode-se dizer também que erradicar – ou pelo menos reduzir – o índice de crimes cibernéticos no Brasil está entre os objetivos do estudo feito ao longo dos experimentos. Afinal, quando se fala em auxiliar pesquisadores e fortalecer estudos a respeito do desempenho em sandboxes de mineração com a plataforma Android, fala-se também em favorecer grupos que trabalham em favor da segurança cibernética desses usuários.

Já quando se refere especificamente à mineração de sandboxes Android, os principais objetivos de pesquisa são:

1. Teste e depuração: sandboxes são usados para testar e depurar aplicativos Android para garantir que eles funcionem corretamente e estejam livres de erros ou falhas.

2. Análise de segurança: sandboxes também podem ser usados para analisar a segurança de aplicativos Android, incluindo detectar vulnerabilidades e atividades maliciosas.

3. Análise de desempenho: sandboxes podem ser usados para medir o desempenho de aplicativos Android, incluindo o seu uso de recursos, velocidade e estabilidade.

4. Teste de compatibilidade: sandboxes podem ser usados para testar a compatibilidade de aplicativos Android com diferentes dispositivos, sistemas operacionais e outros softwares.

5. Mineração de dados: sandboxes também podem ser usados para extrair dados de aplicativos Android para fins de pesquisa ou análise, como o estudo do comportamento do usuário ou a identificação de tendências.

Isso quer dizer que, em linhas gerais, os principais objetivos da mineração de sandboxes Android são melhorar a qualidade, segurança e desempenho de aplicativos Android e reunir dados para fins de pesquisa e análise.

Por fim, e não menos importante, vale ressaltar a importância de avaliar a eficácia do benchmarking proposto como solução de segurança para sistemas Androids. Apesar de ter, como objetivo principal, a busca pela segurança dos usuários, a própria funcionalidade do Droidxp está entre os objetivos avaliados nas discussões deste trabalho.

Capítulo 3

Metodologia

Antes de iniciar a discussão da metodologia utilizada neste trabalho de curso, é importante destacar o contexto em que o mesmo foi desenvolvido. O grupo de pesquisa responsável pelo desenvolvimento deste trabalho foi criado pelo meu orientador e conta com a participação de uma equipe interdisciplinar de profissionais dedicados à investigação de soluções eficientes para o problema de detecção de malware em dispositivos Android.

O grupo de pesquisa em questão tem como objetivo principal apoiar a pesquisa em mineração de sandboxes no sistema Android. Para atingir este objetivo, o grupo tem se dedicado a realizar estudos detalhados sobre as ferramentas existentes de análise dinâmica e estática para detectar ameaças maliciosas em tempo real. A equipe é composta por pesquisadores, professores e alunos com expertise em segurança de dispositivos móveis, análise de malware e programação de sistemas para Android, o que torna o grupo altamente capacitado para enfrentar os desafios apresentados neste campo de pesquisa.

As pesquisas utilizadas neste trabalho de curso podem ser classificadas, predominantemente, por meio do método dialético cujos experimentos trazem diferentes análises e resultados que serão discutidos ao longo das etapas apresentadas na fase de discussão e soluções do problema diagnosticado.

Os experimentos aqui compartilhados contarão com uma abordagem qualitativa, pois avaliam qualitativamente os resultados – em termos de cobertura de código e detecção de malware – do benchmarking proposto. Para isso, foi necessário é necessária uma análise detalhada de malware fora do dispositivo para entender sua funcionalidade.

A análise de amostras pode ser feita manualmente para extrair assinaturas robustas delas. No entanto, dado o rápido aumento de malware, há necessidade de métodos de análise que precisam de intervenção humana mínima, ajudando o analista de malware a gerar uma solução oportuna de novos malwares. Análise estática pode identificar com rapidez e precisão padrões maliciosos, mas falha contra ofuscação de código, bem como execução dinâmica de código no Android.

Assim, abordagens de análise dinâmica, embora demorados, são usados para extrair comportamento malicioso de amostras usando técnicas furtivas, executando-as em um ambiente de caixa de areia.

Durante o desenvolvimento do projeto, minha principal contribuição foi na construção da arquitetura do sistema. Com base nos requisitos do orientador, trabalhei em conjunto com a equipe de desenvolvimento para definir as camadas e componentes necessários para construir um sistema escalável e robusto. Foi um processo iterativo em que tivemos que fazer várias revisões e ajustes para garantir que a arquitetura atendesse às necessidades do projeto.

Também fui responsável pelo desenvolvimento do fluxo de execução inicial do projeto. Trabalhei com a equipe de desenvolvimento para definir as etapas necessárias para a execução do sistema, incluindo a integração de diferentes módulos e a configuração das ferramentas necessárias. Além disso, fui responsável pela execução dos primeiros testes do sistema para verificar se tudo estava funcionando conforme o esperado. Essa etapa foi crucial para identificar problemas e fazer ajustes antes de prosseguir com o desenvolvimento.

Por fim, integrei a ferramenta DroidBot como objeto de comparação no projeto. Isso nos permitiu testar o sistema com diferentes cenários e identificar possíveis problemas em um ambiente controlado. Fui responsável por configurar a ferramenta, definir os casos de teste e executar os testes. Como resultado, conseguimos comparar com sucesso 5 ferramentas.

Capítulo 4

Discussões e Resultados

Para se falar em sistema de segurança e desenvolvimento de benchmark, foi preciso recorrer a estudos de pesquisas anteriores que investigaram a eficácia de sandboxes de mineração, usando ferramentas de teste automatizadas, a fim de comparar o desempenho de ferramentas de geração de teste com sandboxes de mineração. Por exemplo, Bal et al. [9] apresentaram os resultados de um estudo empírico comparando cinco ferramentas de teste automatizado (DroidMate, Monkey, GUIRipper, Puma e Droidbot) para mineração de sandboxes. Os resultados mostram que o DroidBot foi mais eficiente para detectar malware em um conjunto de dados composto por cento e dois pares de aplicativos benignos e malignos [10, 11].

Stoat (STOchastic model App Tester), uma nova abordagem de teste baseada em modelo estocástico para melhorar os testes de GUI de aplicativos Android, tem o objetivo de testar exaustivamente as funcionalidades de um aplicativo a partir do modelo de GUI, validando o comportamento do aplicativo ao impor várias interações de usuário e do sistema. Dado um aplicativo como entrada, o Stoat opera em duas fases. Na primeira delas, ele gera um modelo estocástico do aplicativo para descrever suas interações de GUI.

Um modelo estocástico para um aplicativo é uma máquina de estado finito (FSM) cujas arestas estão associadas a probabilidades de geração de teste. Em particular, Stoat usa uma técnica de análise dinâmica, aprimorada por uma estratégia de exploração de IU ponderada e análise estática, para explorar os comportamentos do aplicativo e construir o modelo estocástico. Em segundo lugar, o Stoat altera iterativamente o modelo estocástico e gera testes a partir das mutações do modelo. Ao perturbar as probabilidades, o Stoat é capaz de gerar testes com várias composições de eventos para testar suficientemente as interacões da GUI e direcionar o teste propositalmente para caminhos menos percorridos para detectar bugs profundos.

Em particular, Stoat usa um algoritmo de pesquisa guiada, inspirado na amostragem Markov Chain Monte Carlo (MCMC), para pesquisar modelos “bons”, espera-se que os

testes derivados sejam diversos, bem como alcancem código alto e cobertura do modelo. Além disso, o Stoat adota uma estratégia simples, mas eficaz para aprimorar o MBT: injetar aleatoriamente vários eventos de nível de sistema em testes de IU durante a amostragem de MCMC. Isso evita a complexidade de incorporar eventos de nível de sistema no modelo de comportamento, e impõe ainda mais a influência do ambiente externo para detectar bugs intrincados.

No que tange às fases em que o Stoat opera, entende-se por "Fase 1" a construção do modelo, ou seja, Stoat primeiro constróii um estocástico Finite State Machine (FSM) para descrever os comportamentos do aplicativo. Para isso, utiliza uma técnica de análise dinâmica, aprimorada por uma exploração de IU ponderada estratégia, para explorar com eficiência os comportamentos do aplicativo. Isso infere eventos de entrada, analisando a hierarquia da IU das páginas do aplicativo, priorizando dinamicamente suas execuções para maximizar a cobertura do código. Além disso, para identificar alguns eventos potencialmente ausentes, um estático (etapa 1) é realizado para verificar os ouvintes de eventos registrados no código do aplicativo. Stoat registra as frequências de execução de todas as IU, eventos durante a exploracão e, posteriormente, utiliza para gerar os valores de probabilidade das transições no modelo.

Entende-se por "Fase 2" a mutação do modelo, geração de teste e execução. Para testar exaustivamente um aplicativo, o Stoat aproveita o modelo da Fase 1 para orientar iterativamente a geração de teste com o intuito de obter alta cobertura, exibindo diversas sequências de eventos. Em detalhes, Stoat funciona como um "loop", ou seja, altera aleatoriamente as probabilidades de transição da corrente modelo estocástico (etapa 4), gera os testes a partir do modelo w.r.t. a probabilidades (etapa 5), eventos de nível de sistema injetados aleatoriamente (analisados por análise estática na etapa 3) nesses testes de nível de IU para aprimorar o MBT (etapa 6), reproduza-os novamente no aplicativo (etapa 7) e colete os resultados do teste, como como cobertura de código e modelo e diversidades de sequência de eventos (etapa 8).

Informado pelos resultados do teste, Stoat explora a amostragem de Gibbs para decidir se o novo modelo proposto deve ser aceito ou rejeitado (passo 9). O modelo com melhor valor objetivo será aceito para a próxima iteração de mutações e amostragens; por outro lado, será rejeitado com certa probabilidade para evitar o ótimo local (se rejeitado, o modelo original será reutilizado). Assim que qualquer bug for detectado (ou seja, falha ou não resposta), uma análise posterior será realizada para diagnosticar o bug com o teste correspondente.

4.1 Background

Sandbox é um ambiente isolado em um dispositivo eletrônico onde aplicativos não podem afetar outros programas ou dados fora de seus limites. Ele permite que sejam realizados testes e execuções de códigos inseguros e não testados – possíveis malwares – sem se preocupar com a integridade do dispositivo eletrônico host. Outro uso é garantir um ambiente de teste não tendencioso, garantindo o mesmo status inicial pré-configurado do ambiente.

Muitos sistemas operacionais de smartphone implementam uma arquitetura de sandbox consolidada para controlar o acesso de aplicativos de terceiros a recursos confidenciais. Como parte desse sandboxing, eles têm um sistema de permissão, que sempre pede aos usuários que concedam ou neguem permissão para acessar esses recursos [12]. O sistema de permissões do Android consiste em quatro tipos de permissões [13]: assinatura e sistema, que são reservados para aplicativos que foram assinados com uma chave disponível para o desenvolvedor do firmware.

Permissões normais, que são concedidas automaticamente ao aplicativo sem o envolvimento do usuário; E as permissões perigosas são apresentadas em um prompt no momento da instalação do aplicativo. Se o usuário continuar com a instalação, as permissões perigosas serão concedidas permanentemente ao aplicativo. Infelizmente, os desenvolvedores Android frequentemente solicitam mais permissões perigosas do que seus aplicativos exigem, causando o problema de aplicativos com privilégios excessivos.

Para resolver esse problema, Jamroziket al. propôs uma nova abordagem, descrita como sandboxes de mineração Minerar Sandboxes Android [14] é uma técnica para extrair regras de sandbox de um determinado aplicativo Android e usar essas regras para garantir a segurança do aplicativo. A técnica traz basicamente duas etapas. A primeira etapa extrai as regras, que de fato irão compor a sandbox, por meio de ferramentas geradoras de testes automáticos. Essas ferramentas exploram o comportamento do programa e monitoram o acesso a API's sensíveis.

Em uma segunda etapa, é levado em consideração que API's não acessadas, ou acessadas de forma diferente da primeira etapa, não devem ser acessadas. Portanto, se um aplicativo malicioso exigir acesso a um novo recurso, diferente do que foi minerado anteriormente, a sandbox irá proibir o acesso a esse novo recurso.

4.2 Princípios de design DroidXP: uma metáfora de benchmarking

O benchmarking é a atividade de medir e avaliar o desempenho relativo de um ativo (por exemplo, um algoritmo ou ferramenta) em relação ao desempenho de outro ativo — considerando condições bem definidas [15]. Portanto, o objetivo é fazer comparações entre diferentes soluções que compartilham um propósito semelhante, ou entre diferentes versões do mesmo Sistema em Teste (SUT).

Para fazer as comparações, alguns instrumentos de medição são necessários, com o intuito de coletar um conjunto de métricas que são relevantes para um determinado domínio. DroidXPfoi projetado para comparar o desempenho das ferramentas de geração de teste com a mineração de sandboxes do Android. De acordo com Bouckaert et al [15]. Pode-se coletar métricas primárias e secundárias durante o benchmarking de ativos computacionais. As métricas primárias são aquelas coletadas diretamente do SUT, e as métricas secundárias são aquelas relacionadas ao ambiente em que o SUT está operando. Em nosso benchmark, estamos mais interessados em dois principais: cobertura de teste e número de malwares detectados.

Em todo e qualquer processo de benchmarking, uma configuração bem definida deve ser uma preocupação fundamental antes de avaliar os SUTs. Por exemplo, no contexto de DroidXP, a configuração corresponde ao corpus de aplicativos Android (pares de aplicativos benignos/maliciosos utilizados na análise), o tempo máximo de execução de cada ferramenta de geração de caso de teste (os sistemas em teste), e o número de repetições nos experimentos. Esses elementos são suficientes para avaliar diversos cenários, que podem subsidiar a pesquisa em área de segurança de mineração. As variáveis de resposta são os meios que usamos para comparar os SUTs. Conforme mencionado, consideramos duas variáveis de resposta: cobertura do teste e número de malwares que cada ferramenta detecta.

Os benchmarkings também levam em consideração outro aspecto importante que traz mais precisão para a atividade de pesquisa: a comparabilidade. Como as ferramentas de geração de teste dependem do não determinismo, as diferentes execuções de uma ferramenta (repetições) nas mesmas configurações podem produzir resultados diferentes (em termos de cobertura de código e detecção de malware). Uma vez que temos que produzir um resultado próximo para comparações, usamos a média da resposta variável coletada em uma série de repetições.

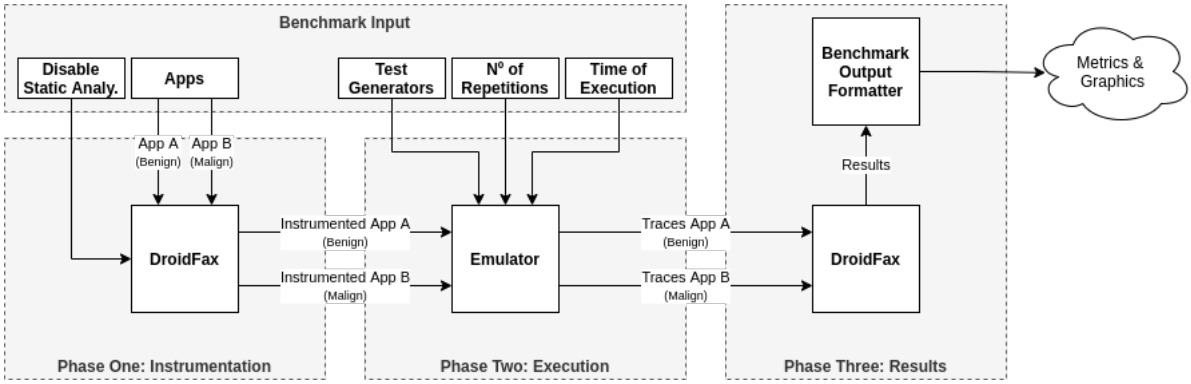

Figura 4.1: Benchmarking architecture. Fonte: elaborado pelo grupo de pesquisa (2021).

4.3 DroidXPbenchmarking

Conforme já citado, as dificuldades para configurar um ambiente que visa integrar ferramentas de geração de testes e reproduzir estudos empíricos em repositórios de software de mineração são os principais motivos para projetar e implementar DroidXP. Para mitigar esses problemas, primeiro conduzimos uma engenharia de domínio para identificar os principais requisitos para usar a ferramenta de geração de casos de teste. Os resultados desta engenharia de domínio, juntamente com os princípios já discutidos, orientaram as decisões de implementação do DroidXP.

Por exemplo, DroidXPdepende de uma simples Interface de linha de comando (CLI) que favorece a execução e configuração do benchmarking. DroidXPtambém depende do DroidFax [16], uma ferramenta que instrumenta aplicativos Android e coleta informações relevantes sobre sua execução (usando as ferramentas de geração de casos de teste). O DroidFax já coleta informações de cobertura de código e o conjunto de API's confidenciais que um determinado aplicativo chama durante a execução de um teste. Para se implementar o DroidXPutilizando a linguagem de programação Python, obteve-se dentre as razões para usar Python: um rico conjunto de bibliotecas para escrever CLI's, para simplificar o processo de chamadas de sistema e para implementar análise de dados; suporte para orientação a objetos construções e reflexão (mecanismos que usamos para criar os pontos de extensão DroidXP); e a familiaridade que os pesquisadores tinham com a linguagem. A CLI DroidXPfornece dois comandos: um comando que lista todas as ferramentas de geração de casos de teste (executando o projeto com a opção" list-tools") que foram integradas ao benchmark; e um comando que realiza a execução do benchmark, que pode ser configurado usando os seguintes parâmetros

- tools: especifica as ferramentas de teste usadas no experimento
- t: especifica o limite (em segundos) para o tempo de execução no experimento

- r: especifica o número de repetições usadas na experiência
- output-format: especifica o formato de saída
- debug: especifica a execução no modo DEBUG (padrão: falso)

A arquitetura do DroidXPvem do estilo arquitetônico de tubos e filtros [17] (Figure 4.1), além de incluir 3 componentes principais onde cada um deles é responsável por uma fase específica do benchmarking – instrumentação, execução e análise de resultados.

4.4 Fase 1: Instrumentação

Na primeira fase, um pesquisador deve definir o conjunto de arquivos APK que o DroidXP-deve considerar durante a execução de benchmark. Depois disso, o DroidXPinicia o serviço de instrumentar cada arquivo APK, de modo que seja capaz de coletar dados sobre cada execução. Para melhorar o desempenho do benchmark, a fase de instrumentação é executada apenas uma vez para cada APK. Nesta fase, a ferramenta DroidFax também executa alguns procedimentos de análise estática, com o intuito de coletar a quantidade de métodos e classes de cada aplicativo, uma informação necessária para estimar a cobertura do código.

4.5 Fase 2: Execução

Nesta fase, o DroidXPinstala um arquivo APK (já instrumentado) em um emulador Android e, em seguida, executa uma ferramenta de geração de caso de teste durante um intervalo de tempo. Este processo se repete para cada geração de caso de teste, ferramenta e arquivos APK. Para fornecer repetibilidade do experimento, o DroidXPremove todos os dados armazenados no emulador antes de começar uma nova execução. Ou seja, toda execução usa um emulador fresco, sem qualquer informação que possa ter sido mantida durante experimentos anteriores. O benchmark foi projetado para que facilmente pudesse adicionar um novo teste ferramentas de geração de casos. Para atingir esse objetivo, foi alavancado o Strategy Design pattern [18], o que estabelece um contrato entre uma família de classes que, em nesse caso, abstrai as especificidades para executar cada ferramenta que se deseja integrar em DroidXP. De acordo com este contrato, deve-se: implementar uma classe que herda de `ToolSpec`; definir um construtor que chame o construtor `super`, passando o nome, a descrição e o ID do processo como argumentos; e implementar o método `execute`, que recebe como argumento o caminho de um arquivo APK e um tempo limite.

Listing 1. Amostra ToolSpec

```
class ToolSpec(AbstractTool):
  def_init__(self):
    super(ToolSpec, self)._init_(
      "TestGeneratorName",
      "TestGeneratorDescription",
      "process_id"
    )

  def execute(self, path, timeout):
    #Executetestgenerator...
```

Figura 4.2: Fonte: elaborado pelo grupo de pesquisa (2021).

DroidXPEle usa o último parâmetro para encerrar o processo de execução no emulador após o lançamento de um evento de tempo limite. Esta etapa é necessária para fornecer um ambiente novo para a próxima execução de teste. A lógica real para executar uma ferramenta específica reside no método, portanto, é responsabilidade de um desenvolvedor definir todas as configurações necessárias para executar um determinado ferramenta de geração de casos de teste. Já integramos as seguintes ferramentas em DroidXP.

- Monkey é um utilitário de teste do Google que gera luxos pseudoaleatórios de eventos do usuário.
- DroidBot é um gerador de entrada de teste para Android, que envia eventos de entrada aleatórios ou com script [19].
- DroidMate é um gerador de execução automatizado mecanismo de análise dinâmica para aplicativos Android [20].
- Sapienz é uma abordagem multi-objetivo para teste de entrada automatizado geração para Android [21].
- Humanoid é uma ferramenta que usa técnicas de aprendizado profundo para explorar aplicativos Android, imitando o comportamento humano [11].
- Stoat (STochastic model App Tester) é uma abordagem guiada para realizar testes baseados em modelos estocásticos em aplicativos Android. A ideia é testar minuciosamente as funcionalidades de um aplicativo a partir de seu modelo GUI e validar o comportamento do aplicativo aplicando várias interações usuário/sistema.

4.6 Fase 3: Análise de resultados

Durante a execução, todos os dados necessários para calcular os resultados são fornecidos pelo Logcat [22] – uma das ferramentas nativas do Android SDK de linha de comando, que despeja um log do emulador Android. Assim, a única parte do log que é analisada nesta fase são as mensagens enviadas pelos métodos dentro do aplicativo Android que foram instrumentadas na primeira fase com a ferramenta DroidFax. O Droidfax calcula a cobertura alcançada por cada teste e qual API sensível foi acessada durante a execução desse teste. Essa última informação é necessária para calcular o desempenho do gerador de teste na identificação de aplicativos maliciosos, identificando diferenças entre a API sensível acessada por cada versão de um aplicativo. Essas informações são vitais para a medição do desempenho e qualificação do gerador de teste. Após essa fase, o benchmark produz os resultados do experimento, que informa o desempenho de uma ou mais ferramentas geradoras de teste em sandboxes de mineração.

4.7 Estudo Empírico

4.7.1 Configurações de estudo

Este estudo empírico visa reproduzir um anterior trabalho de pesquisa em mineração de sandboxes [9], além de ter, como objetivo, o experimento de uso do DroidXP. Semelhante ao estudo anterior, aqui também foram aproveitadas duas métricas para comparar o desempenho das ferramentas de geração de casos de teste: código cobertura e o número de malware detectado.

Para cobertura de código, foi utilizada a porcentagem de métodos de aplicação em cada ferramenta percorrida — durante a fase de execução. Além disso, também foram utilizados os mesmos pares de aplicativos da versão do estudo anterior — um corpus que contém uma amostra de 102 pares (benignos /malignos) de aplicativos de AndroidZoo [23]. Foram investigadas as seguintes questões por meio do estudo:

> [(RQ1)]Qual é o desempenho de cada ferramenta em termos do número de malware detectado? Qual é a foça da correlação entre a cobertura de código e o número de malware detectado? Qual é a relação entre a taxa de cobertura e a precisão de cada ferramenta na detecção de aplicativos maliciosos?

Semelhante ao trabalho de Bao et al. [9], foram dois experimentos conduzidos. No primeiro, houve execução de um benchmark considerando todos os 102 aplicativos em nosso corpus, e um tempo limite de execução de um minuto. O primeiro experimento foi

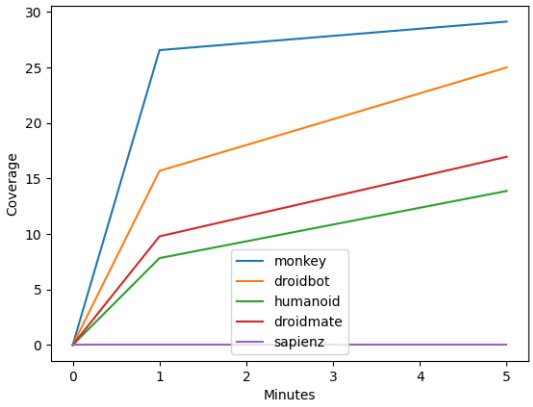

Figura 4.3: Coverage: 1 and 5 minutesGráfico, Gráfico de linhas

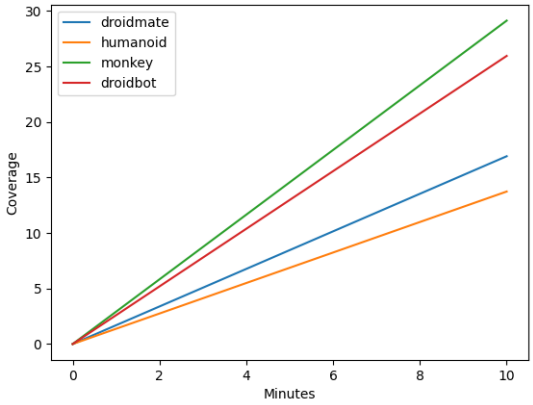

Figura 4.4: Coverage: 10 minutes

utilizado para responder a primeira questão de pesquisa (RQ1). No segundo, foi considerado um subconjunto dos aplicativos no corpus, compreendendo apenas 10 aplicativos; e executou o benchmark em três configurações diferentes de tempo limite: 1 minuto, 5 minutos e 10 minutos. O segundo experimento foi utilizado para responder às questões RQ2 e RQ3.

4.8 Resultados do estudo

A Figura 4.6 resume as descobertas feitas com acerca da primeira questão de pesquisa – todas as ferramentas foram capazes de identificar corretamente pelo menos 30% dos malwares. O curioso é que no discorrer dos estudos, duas ferramentas recentes obtiveram o melhor desempenho: Sapienz (54%) e Humanóide (42%). Essas ferramentas não foram consideradas no trabalho anterior. No entanto, percebemos um menor desempenho das

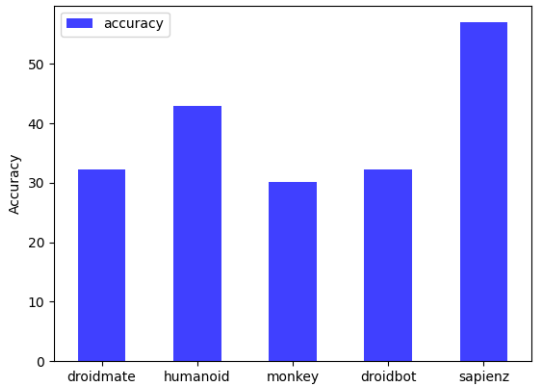

Figura 4.5: Accuracy: 96 pairs Apps

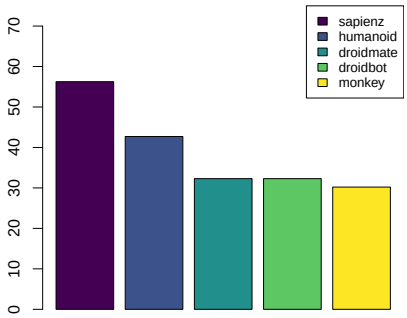

Figura 4.6: Resumo da porcentagem de malwares detectado corretamente

ferramentas DroidMate, DroidBot e Monkey; em comparação com os resultados do estudo anterior [9]. O intuito é dar continuidade a uma investigação acerca das possíveis razões que justifiquem essa diferença.

A Figura 4.7 resume as descobertas feitas abordando a segunda questão de pesquisa (RQ2). Percebemos duas descobertas interessantes: primeiro, Monkey e Humanoid superam as ferramentas restantes em termos de cobertura de código, ao executar nas duas primeiras configurações: *Config-(a)* (1 minuto) e *Config-(b)* (5 minutos). Surpreendentemente, Humanoid levou a uma cobertura de código menor para *Config-(c)* (10 minutos) do que para *Config-(b)*. Além disso, não percebemos nenhum benefício de aumentar o tempo limite de cinco a 10 minutos, ao considerar os critérios de cobertura de código.

Por fim, foi possível responder à pergunta de pesquisa (RQ3), usando o teste de correlação de Spearman. Dessa maneira, foi estimada a força da correlação entre a cobertura do método e número de malwares detectados. Os resultados revelam uma correlação mo-

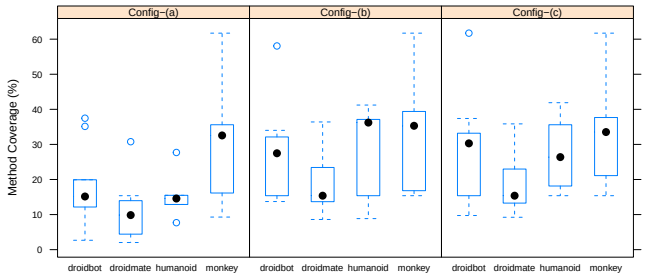

Figura 4.7: Resumo da cobertura do método sobre as ferramentas

derada (p-valor = 0,68), negativa (= 0,31) entre essas medições. Ou seja, melhorando o desempenho da cobertura de código pode não diretamente contribuir para o desempenho de detecção de malware.

4.9 Discussão e limitações

Este estudo empírico possibilitou um experimentar bem-sucedido de integração do DroidXP- com seis ferramentas diferentes, o que embasou as pesquisas anteriores [9]. No entanto, alguns resultados diferem do trabalho de Bao et al. [9] e isso pode ter ocorrido devido às diferenças de reprodução quanto às configurações da pesquisa anterior, sem descartar ainda a possibilidade de o DroidXPter impactado, de alguma forma, o desempenho das ferramentas. Com isso, faz-se necessário investigar o problema com profundidade em um momento futuro.

Uma outra limitação diagnosticada no estudo é que DroidXPnão coletou a métrica de cobertura de código para a ferramenta Sapienz – o que não deixou claro a origem do problema. Contudo, não foi possível coletar a cobertura do Sapienz, mesmo ao executá-lo separadamente de DroidXP.

A ferramenta Puma não pôde ser adicionada ao benchmarking devido à desatualização de suas dependências. Isso causou problemas na hora da integração com o DroidXP, impossibilitando a coleta de métricas precisas para a avaliação da eficácia da ferramenta. Além disso, é importante lembrar que o benchmarking precisa incluir ferramentas atualizadas e em pleno funcionamento para garantir a validade dos resultados obtidos. Portanto, é recomendável procurar outras ferramentas que possuam dependências atualizadas e funcionem adequadamente com o DroidXP.

Além disso, levantou-se ainda uma outra fragilidade importante de ser aperfeiçoada no desenvolvimento dessa ferramenta, que foi quanto à agilidade no processo de verificação de segurança do aplicativo. Para isso, foi iniciado um trabalho de pesquisa e testes a respeito

de uma ferramenta que poderia atuar como parceira, por meio de um plug-in. Contudo, como esse trabalho foi apenas iniciado, vale ressaltar a necessidade de aprofundar no assunto – funcionalidade da ferramenta, como funciona e quais as perspectivas dela – na próxima seção deste trabalho ("trabalhos futuros"), que foi dividido desta etapa para que sirva como norteador de outros estudantes da área de Ciências da Computação (ou pesquisadores) que decidam dar continuidade aos experimentos da DroidXP, aperfeiçoando suas funcionalidades.

4.10 Trabalhos relacionados

Até onde sabemos, este é o primeiro trabalho que propôs uma ferramenta de Benchmark para geração de casos de teste. No entanto, muitos trabalhos apresentam um estudo comparativo das principais ferramentas de geração de entrada de teste existentes para o Android [24][9]. Além disso, outros artigos [19][11][25][20][21] fornecem um conjunto de geradores de entrada automatizados que podem ser medidos e analisados por nossa ferramenta proposta. Assim, Li et al. propuseram em [19] o DroidBot, um gerador de entrada de teste guiado por interface do usuário para aplicativos Android, que suporta a geração de entrada de teste baseada em modelo. Neste artigo, os autores mostram a comparação entre o DroidBot e o Monkey[26] em um exemplo de prova de conceito de uso do DroidBot na análise de malware, que poderia ter sido facilmente realizado em nossa ferramenta de benchmark. Dos mesmos autores, outro gerador de entrada de teste GUI para Android, descrito como Humanoid [11], é uma evolução do Droidbot e apresenta uma proposta capaz de gerar entradas de teste semelhantes a humanos, usando aprendizado profundo. Mesmo como o artigo DroidBot, este trabalho apresenta comparações entre ferramentas de teste de ponta e apresenta resultados de cobertura de teste mais altos. [25] Propôs Stoat, outra geração de entrada de teste baseada em modelo. O artigo apresenta comparações entre outras 3 ferramentas, expondo a cobertura de linha, e estatísticas de erros e falhas, mas não se concentra nas capacidades de detecção de malware.[21] Apresenta o Sapienz, uma ferramenta de geração de entrada de teste que realiza análise estática e dinâmica e realiza um estudo de benchmark entre outras 2 ferramentas, apresentando falhas detectadas e taxa de cobertura. Todos esses estudos são possíveis de serem reproduzidos no DroidXP, que possui uma integração mais fácil com outras ferramentas que surgem.

Capítulo 5

Recomendações e trabalhos futuros

Com isso, abriu-se a possibilidade um trabalho futuro para uma pesquisa de benchmarks de detecção de malware em sistemas Android com a utilização de uma ferramenta chamada Java-MOP, a fim de especificar e monitorar propriedades de segurança durante a execução de aplicativos Android., o que poderia permitir a detecção de violações de segurança e identificação de comportamentos maliciosos em tempo real.

Uma das possíveis abordagens seria utilizar o Java-MOP para escrever especificações de segurança que identifiquem o comportamento desejado, incluindo regras para garantir que o aplicativo não acesse dados confidenciais sem autorização, não envie dados sensíveis para servidores remotos sem autorização e não realize ações maliciosas, como instalar outros aplicativos ou enviar SPAM – prática de envio de mensagens automáticas não solicitadas ou desejadas pelo receptor – por exemplo. Para que isso seja possível, a sugestão é testar como essa ferramenta se comporta em termos de otimização, se for plugada à DroidXP.

Adicionando o JavaMOP na fase de execução do DroidXP, cogitou-se a possibilidade de agilizar o tempo de duração das verificações de segurança, otimizando significativamente a eficiência da ferramenta. E devido à possibilidade de tornar a ferramenta de benchmarking ainda melhor, potencializando não somente sua eficiência, mas em ganho de agilidade também, foi que surgiu então a ideia de iniciar os testes e analisar o resultado dessa segunda etapa do experimento.

Uma outra possível abordagem seria a utilização de técnicas de análise dinâmica, como a técnica de emulação dinâmica (simulação que consiste em testar a ferramenta para avaliar a previsibilidade de como ela se comportaria), combinada com o Java-MOP, para monitorar a execução do aplicativo e verificar se ele viola as especificações de segurança. Isso poderia permitir, por exemplo, a detecção de malware desconhecido, que não seria detectado por mecanismos de detecção baseados em assinatura. Portanto, para um trabalho futuro, torna-se uma alternativa.

Além disso, é possível utilizar técnicas de aprendizado de máquina, como redes neurais,

para treinar o Java-MOP a reconhecer comportamentos maliciosos. Isso poderia melhorar a precisão da detecção de malware e ser usado para identificar comportamentos maliciosos que ainda não foram documentados.

Em resumo, o Java-MOP pode ser usado em trabalhos futuros para uma pesquisa de benchmarkings de detecção de malware em sistemas Android, especificando e monitorando propriedades de segurança durante a execução de aplicativos Android, permitindo a detecção de violações de segurança e identificação de comportamentos maliciosos em tempo real, combinando técnicas de análise dinâmica, aprendizado de máquina e emulação dinâmica, para melhorar a precisão e eficácia da detecção de malware.

Em função do tempo exigido por esses testes não ter atendido suficientemente o tempo exigido para a execução das etapas de aplicação, observação, avaliação e discussões, fez-se necessária a inclusão deste guia de recomendações para trabalhos futuros. Assim, ficam as indicações e orientações aos estudiosos – e até mesmo autores deste trabalho – que manifestem interesse em dar continuidade aos experimentos que visem cumprir os objetivos associados às pesquisas apresentadas.

Capítulo 6

Considerações finais

Desde o início do Android, existem pesquisadores interessados em encontrar soluções de segurança contra-ataques de malware no sistema Android, que se popularizou e apresenta vulnerabilidades de segurança. Este trabalho buscou contribuir com esses estudos, por meio do uso da ferramenta DroidXP.

DroidXP foi criado para ajudar os pesquisadores a desenvolver novas soluções de ferramentas de teste e análise, e seus resultados foram positivos na construção de sandboxes melhores. No entanto, foram identificadas algumas fragilidades na ferramenta, como a limitação na coleta da métrica de cobertura de código para a ferramenta Sapienz e a agilidade de processamento, que poderia ser melhorada com o uso de outros aplicativos.

Por fim, foi sugerido o uso do Java-MOP como plug-in da ferramenta para sanar as vulnerabilidades encontradas, mas isso exigiria mais tempo do que o disponível para este trabalho. Conclui-se que ainda há muito a ser feito para encontrar soluções eficazes de segurança contra ataques de malware no sistema Android, e que a ferramenta DroidXP é um passo importante nesse sentido. No entanto, vale ressaltar que alguns dados dos resultados apresentados neste trabalho foram atualizados e continuarão em evolução, pois é necessário continuar explorando novas abordagens e otimizando as existentes para oferecer uma proteção adequada aos dados dos usuários.

Referências

[3] *Android hipposms*. http://www.csc.ncsu.edu. Acesso em janeiro de 2023. 1

[2] *Android: a longa e doce história do sistema operativo da google*. https://www.rptech.radiopopular.pt. Acesso em janeiro de 2023. 1

[3] *O que é o sistema android?* https://edu.gcfglobal.org/pt/como-usar-o-sistema-android. Accessed: August 2022. 1

[4] Faruki, Parvez, Arshad Bharmal, Vijay Laxmi, Vaibhav Ganmoor, Manoj Singh Gaur, Mauro Conti e Muttukrishnan Rajarajan: *Android security: A survey of issues, malware penetration, and defenses*. IEEE Communications Surveys & Tutorials, 17(2):998–1022, 2014. 1, 2

[5] Tan, Darell JJ, Tong Wei Chua e Vrizlynn LL Thing: *Securing android: a survey, taxonomy, and challenges*. ACM Computing Surveys (CSUR), 47(4):1–45, 2015. 2

[6] *O que é segurança da informação?* https://tecnoblog.net. Accessed: December 2022. 2

[7] *Os tipos de malware mais perigosos para android*. https://www.securityreport.com.br. Accessed: January 2023. 2, 3, 4, 5

[8] *Os pilares da segurança da informação*. https://senhasegura.com. Accessed: December 2022. 4, 6

[9] Bao, Lin, Tien Duy B Le e David Lo: *Mining sandboxes: Are we there yet?* Em *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, páginas 445–455. IEEE, 2018. 13, 20, 22, 23, 24

[10] Cai, Haoyu e Barbara G Ryder: *A longitudinal study of application structure and behaviors in android*. IEEE Transactions on Software Engineering, 2020. 13

[11] Li, Yuanchun, Ziyue Yang, Yao Guo e Xiangqun Chen: *A deep learning based approach to automated android app testing*. arXiv preprint arXiv:1901.02633, 2019. 13, 19, 24

[12] Au, Kevin WY, Yajin Zhou, Zhenyu Huang e David Lie: *Pscout: analyzing the android permission specification*. Em *Proceedings of the 2012 ACM conference on Computer and communications security*, páginas 217–228. ACM, 2012. 15

[13] Au, Kevin WY, Yajin Zhou, Zhenyu Huang, Philip Gill e David Lie: *Short paper: a look at smartphone permission models*. Em *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, páginas 63–68. ACM, 2011. 15

[14] Jamrozik, Konrad, Philipp von Styp-Rekowsky e Andreas Zeller: *Mining sandboxes*. Em *Proceedings of the 38th International Conference on Software Engineering*, páginas 37–48, 2016. 15

[15] Bouckaert, Stefan, JVV Gerwen, Ingrid Moerman, Stephen C Phillips e Jerker Wilander: *Benchmarking computers and computer networks*. EU FIRE White Paper, 2010. 16

[16] Cai, Haoyu e Barbara Ryder: *Understanding application behaviours for android security: A systematic characterization*. Relatório Técnico, Department of Computer Science, Virginia Polytechnic Institute & State, 2016. 17

[17] Buschmann, Frank, Regine Meunier, Hans Rohnert, Peter Sommerlad e Michael Stal: *Pattern-Oriented Software Architecture - Volume 1: A System of Patterns*. Wiley Publishing, 1996, ISBN 0471958697. 18

[18] Gamma, Erich, Richard Helm, Ralph Johnson e John Vlissides: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman Publishing Co., Inc., USA, 1995, ISBN 0201633612. 18

[19] Li, Yuanchun, Ziyue Yang, Yao Guo e Xiangqun Chen: *Droidbot: a lightweight ui-guided test input generator for android*. Em *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, páginas 23–26. IEEE, 2017. 19, 24

[20] Jamrozik, Konrad e Andreas Zeller: *Droidmate: a robust and extensible test generator for android*. Em *Proceedings of the International Conference on Mobile Software Engineering and Systems*, páginas 293–294, 2016. 19, 24

[21] Mao, Ke, Mark Harman e Yue Jia: *Sapienz: Multi-objective automated testing for android applications*. Em *Proceedings of the 25th International Symposium on Software Testing and Analysis*, páginas 94–105, 2016. 19, 24

[22] *Logcat*. `http://developer.android.com/tools/help/logcat`. Accessed: March 2020. 20

[23] Allix, Kevin, Tegawendé F Bissyandé, Jacques Klein e Yves Le Traon: *Androzoo: Collecting millions of android apps for the research community*. Em *Proceedings of the 13th International Conference on Mining Software Repositories*, páginas 468–471. ACM, 2016. 20

[24] Choudhary, Shauvik Roy, Alessandra Gorla e Alessandro Orso: *Automated test input generation for android: Are we there yet?(e)*. Em *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, páginas 429–440. IEEE, 2015. 24

[25] Su, Ting, Guozhu Meng, Yuting Chen, Ke Wu, Weiming Yang, Yao Yao, Geguang Pu, Yang Liu e Zhendong Su: *Guided, stochastic model-based gui testing of android apps*. Em *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, páginas 245–256, 2017. 24

[26] *Monkey*. `https://developer.android.com/studio/test/monkey`. Accessed: February 2020. 24