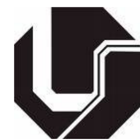
CEEL - ISSN 2596-2221
Universidade Federal de Uberlândia
06 a 10 de Dezembro de 2023



TAXONOMIA DE MALWARES DE DISPOSITIVOS MÓVEIS

Fabio Neves Rezende

FACOM - UFU - Universidade Federal de Uberlândia

Kil Jin Brandini Park

FEELT - UFU - Universidade Federal de Uberlândia

***Resumo* - Com o recente crescimento do mercado "mobile", surgiram também os malwares para dispositívos móveis. Estes programas maliciosos evoluiram muito em complexidade e dificuldade em serem detectados. Este artigo tem como objetivos elucidar os passos necessários para se realizar a inspeção de aplicativos Android para detecção de programas que causem danos tanto ao dispositivo quanto à integridade, confidencialidade ou disponibilidade dos dados do usuário e utilizar de uma nomenclatura mais concisa que a usual para classificar estes malwares de forma taxonômica. Serão apresentados quatro casos de malwares analisados segundo as técnicas descritas neste trabalho.**

***Palavras-Chave*- Sistema Operacional Android, Mobile, Análise de Malware. Taxonomia de Malware.**

MOBILE DEVICES MALWARE TAXONOMY

***Abstract* - With the recent growth of the "mobile" market, malware for mobile devices have also emerged. These malicious programs have evolved a lot in complexity and difficulty in being detected. This article aims to elucidate the necessary steps to carry out the inspection of Android applications to detect programs that cause damage both to the device and to the integrity, confidentiality or availability of users data and to use a more concise than usual nomenclature to classify these malware. Four cases of malware analyzed according to the techniques described in this work will be presented.**

***Keywords* - Android Operating System, Mobile, Malware Analysis, Malware Taxonomy.**

NOMENCLATURA

APK Android Package.

BHO Browser Helper Object.

DEX Dalvik Executable.

IM Instant messaging.

IRC Internet Relay Chat.

PAC Proxy auto config.

I. INTRODUÇÃO

Nos últimos anos o mercado experimentou uma expansão nas tecnologias computacionais focadas nos aparelhos móveis, muitas das quais foram desenhadas para computadores fixos como desktops necessitando, portanto, de adaptações em suas implementações. Muitas vezes a segurança tornou-se um fator secundário em oposição à satisfação do mercado, fato gerador de novas vulnerabilidades. O objetivo deste trabalho é apresentar uma metodologia de engenharia reversa de programas para o sistema operacional Android, pois estes são os mais populares no mercado atualmente [3]. Além disso, no ano de 2019 este sistema operacional foi considerado o mais vulnerável de todos [4]. Após a aplicação da análise, os aplicativos serão classificados taxonomicamente de acordo com seu funcionamento. De inicio, será apresentado uma metodologia que consiste em entender, tecnicamente, como é a estrutura do sistema operacional em questão e os programas que funcionam nele. Então serão apresentados os passos necessários para se analisar em ambiente controlado o comportamento dos malwares. Para isto, serão utilizados samples (amostras) com o fim de se exemplificar o uso desta metodologia. Finalmente, tais programas maliciosos serão classificados usando um método taxonômico com o intuito de mitigar as ambiguidades causadas pelos modos atuais de nomear softwares maliciosos [2]. O foco deste trabalho é de examinar e classificar alguns malwares seguindo uma maneira estruturada de nomenclatura dos programas maliciosos proposta por [2]. A maneira usual de nomeação (usando nomes como por exemplo Virus, Worm, trojan, etc) apresenta falta de objetividade para estabelecer o relacionamento malware - comportamento. A ideia proposta por [2] é estabelecer uma nova maneira de se nomear os programas maliciosos de forma a se ter uma visão macro do que ele faz baseado no nome da sua classificação e também de se identificar certos padrões de comportamento que indicam atividade maliciosa. Nos resultados levantados pelo estudo supracitado é possível identificar comportamento malicioso até mesmo quando antivirus convencionais não o fazem. É importante ressaltar que o trabalho de [2] foi feito para programas que rodam no sistema operacional Windows XP. No caso deste artigo, serão feitas as modificações necessárias e

considerações que se aplicam ao universo do sistema operacional Android. Este trabalho classifica os malwares segundo seu comportamento da seguinte maneira:

Tabela 1: Classificação dos Malwares

Classe	Comportamento	Rótulo	Processo de identificação
Evasão	Remoção de evidências	RE	Amostra deleta a si ou artefatos relacionados
	Remoção de registros	RR	Amostra deleta chaves para burlar o Safe-boot
	Terminação de antivirus	TA	Amostra finaliza processos de antivirus
	Terminação de firewall	TF	Amostra encerra execução de firewall
	Remoção de avisos de atualização	TU	Amostra desliga avisos de atualização
Perturbação	Escaneamento de serviços vulneraveis	VS	Amostra realiza escaneamento de rede nas portas 135,445
	Envio de emails	ES	Amostra tenta enviar emails ou spams pela rede
	Conexão em porta de IRC/IM	IP	Amostra se conecta em portas e serviços de IRC/IM
	Comandos de IRC/IM não encriptados	IC	Amostra envia comandos de IRC/IM (como NICK,JOIN) na rede
Modificação	Criação de novo binário	NB	Amostra cria arquivo binário ou modifica binário previamente existente
	Edição de binários do sistema	CB	Amostra edita arquivo binário local
	Criação de Mutex não usual	UM	Amostra cria mutex não presente na whitelist
	Modificação de arquivo de DNS	HC	Amostra modifica o arquivo 'hosts.txt'
	Modificação de proxy de navegador	PL	Amostra registra um arquivo/local PAC
	Modificação de comportamento de navegador	BI	Amostra registra um BHO
	Persistência	PE	Amostra se adiciona a chave de registro 'Run'
	Download de malware conhecido	DK	Amostra baixa arquivo e o submete a teste de antivirus para verificar se é detectado
	Download de arquivo desconhecido	DU	Amostra baixa arquivo que não conseguimos classificar
	Carregamento de drivers	DL	Amostra tenta carregar um arquivo .sys
Roubo	Roubo de dados do sistema ou usuário	IS	Amostra lê um dado do usuário e tenta o enviar pela rede
	Roubo de dados credenciais ou financeiros	CS	Amostra obtém credenciais ou informações financeiras
	Sequestro de processo	PH	Amostra escreve em espaço de memória de outro processo

Fonte: [2]

É evidente que não será possível seguir a risca a classificação acima pois a mesma foi feita para malwares do sistema Windows. Considerando que a análise será feita em aplicativos para Android, algumas modificações são pertinentes. Por exemplo, os escaneamentos citados no rótulo VS podem se aplicar também a outras portas que o Android ou seus serviços usam. O arquivo de resolução de nomes de domínio no Linux é o /etc/hosts e não o hosts.txt. E assim por diante dentre outras modificações pertinentes. Isto não foi citado na tabela acima para que a mesma fique equivalente ao trabalho original alterando apenas o idioma de inglês para português. Para prosseguir com a análise dos programas tem-se como base auxiliar trabalhos como o livro [5] e também nas documentações oficiais dos código utilizadas ao longo da análise. Para referências ao sistema operacional Android, será utilizado [6]. Esta fonte contém tudo que é necessário em relação à documentação do sistema operacional, suas chamadas de API, exemplos de uso, entre outros. Considerando que muitos malwares se utilizam de técnicas de ofuscação para dificultar a ação de engenharia reversa, também serão usadas algumas ténicas citadas em [7] na hora de procurar por binários e/ou informações importantes do malware que estejam escondidas. Este trabalho lista as técnicas mais comuns de ofuscação de código e como detectá-las, enfatizando que a ofuscação de código é capaz de esconder códigos maliciosos da maioria dos antivirus comerciais existentes até a data de sua publicação. Também será usado como guia o manual produzido por [1], dado que este apresenta uma metodologia profissional para engenharia reversa de aplicativos e também para testes de segurança de sistemas e aplicativos Android. A seguir, apresenta-se uma tabela com as modificações criadas para portabilizar as classificações de comportamento de malware para um cenário apropriado, considerando o sistema operacional Android.

Tabela 2: Classificações específicas para android

Classe	Comportamento	Rótulo	Processo de identificação
Modificação	Criação de arquivo não binário	MB	Amostra cria um arquivo não executável
Roubo	Leitura de mensagem SMS	RSM	Amostra lê mensagens de SMS
Perturbação	Envio de mensagem SMS	SSM	Amostra envia mensagem de SMS

Fonte: De autoria própria

Considerando que é comum várias aplicações maliciosas para Android acessarem serviços de celular e telefonia como por exemplo o Serviço de Mensagens Curtas (SMS) e outros. Também foi adicionado o rótulo MB para um dos exemplares analisados adiante.

II. METODOLOGIA

A seguir apresenta-se o básico da estrutura do SO Android necessário e suficiente para o entendimento do processo de engenharia reversa de aplicativos proposta a ser aplicada durante o processo de análise.

A. Estrutura do sistema operacional Android

O Android é um sistema operacional baseado no Kernel Linux [6]. Na camada mais próxima do hardware encontram-se por exemplo os drivers de dispositivo. A diferença do Kernel Linux para o Android é que no segundo constam algumas adições focadas para o mundo Mobile (dentre elas o Low Memory Killer que é o gerenciador de memória com uma política mais agressiva quanto a preservação do espaço da memória disponível) [6]. Logo acima do kernel encontra-se a camada HAL (hardware abstraction layer) composta de um conjunto de funcionalidades para os serviços do SO Android acessarem o kernel sem ter que diretamente efetuar chamadas de sistema (syscalls). Vale lembrar que isso não impede de se acessar a syscall diretamente. Logo acima tem-se a camada que é chamada de "nativa". Nela encontram-se a máquina virtual do Android (previamente chamada de Dalvik, hoje de ART - Android Runtime) e também no mesmo nível as bibliotecas escritas em C/C++ (libc por exemplo) que são usadas pelas

APIs em Java que rodam logo acima. Acima da camada nativa tem-se a camada Java provedora de conteúdo e serviço para os aplicativos. Esta camada provê serviços básicos como Gerenciadores de Activities, localização, notificação, telefonia, display, dentre outros. Finalmente, rodando em cima da camada provedora de serviços tem-se os aplicativos que consomem os serviços providos pela camada inferior.

B. Estrutura de um APK

O arquivo de extensão APK é usado para se instalar um aplicativo no Android. Ele é basicamente um arquivo compactado que contém os recursos (por exemplo imagens e configurações) e o código do programa. Ele pode conter também bibliotecas estáticas compiladas para código nativo que a aplicação usa. Os arquivos e pastas que sempre estarão presentes no APK são: META-INF, res, AndroidManifest.xml, classes.dex. A pasta META-INF contém as informações de assinatura da aplicação com um certificado. São arquivos usados para validar quem é o emissor da aplicação. A pasta res contém os recursos usados pelo aplicativo e pode incluir imagens, áudios, vídeos, os arquivos XML que referenciam componentes visuais providos pela API Java do android de componentes visuais (activity manager, por exemplo), dentre outros. Vale ressaltar que os arquivos XML vêm compactados no formato binário e é suficiente o uso da aplicação APKTOOL para passá-los para modo texto. O arquivo AndroidManifest.xml (chamado de manifesto da aplicação) contém diretivas gerais sobre a execução da mesma. Isto inclui o nome da aplicação e sua versão, os componentes que a aplicação utiliza, as permissões que a aplicação necessita (exemplo: permissão para acessar a câmera, o microfone, etc), ponto de partida da execução (qual classe e método terão o papel de "função main" ao se iniciar o programa), os requisitos de hardware necessários (quantidade mínima de memória, processamento, etc) [6]. Finalmente, o arquivo classes.dex é o arquivo que contém o código bytecode que é interpretado pela máquina virtual do Android (Dalvik ou Rrt). Será dito mais sobre o DEX na proxima sessão.

C. De Java ao Smali e Dex

DEX trata-se de uma abreviação de Dalvik Executable que, apesar do nome, também é executado na Android runtime. O DEX é basicamente um arquivo escrito no formato binário que contém código que é interpretado e executável pela máquina virtual do Android (que é uma Java VM). O arquivo Smali é um arquivo de texto que contém o código "bytecode assembly" formado quando se converte o código de baixo nível do DEX para um formato textual que pode ser mais facilmente lido por humanos e se assemelha à linguagem assembly, porém com suas nuances particulares de sintaxe. Pode-se gerar arquivos smali tanto com o apktool ao se descompactar o APK ou diretamente pelo programa baksmali [12].

O programa enjarify [9] da Google junto com jd-gui [10] são os programas utilizados para se converter código DEX em código Java que é mais facilmente lido por humanos. O primeiro converte o DEX em arquivos .class que contém as definições de classes e o segundo lê estes .class e escreve um código Java equivalente. No entanto, isto não impede de se realizar a engenharia reversa diretamente pelos arquivos Smalis gerados do DEX pois também apresentam uma linguagem relativamente amigável de ser lida. É aconselhável utilizar o enjarify ao invés do dex2jar pois o segundo é uma ferramenta mais antiga e que pode gerar códigos não exatos em alguns casos [9].

III. ENGENHARIA REVERSA NA PRÁTICA

A. Preparo do ambiente controlado

Para ser realizada a engenharia reversa, tem-se primeiro que preparar o ambiente e instalar o que é necessário para o trabalho. Primeiro, tem-se que instalar o Android Studio pois ele contém várias ferramentas que são necessárias, dentre elas o Android SDK, o emulador Android e o depurador de aplicações. Basta seguir ao site oficial (https://developer.android.com/studio/) baixar a aplicação e seguir os passos de instalação lá descritos. Tendo o Android Studio instalado, tem-se agora que definir uma máquina virtual (emulador) para a aplicação ser executada. A forma mais simples de se fazer isso é, dentro do Studio, abrir a opção "AVD Manager" (que fica na barra superior) e clicar em "Criar dispositivo virtual". O dispositivo virtual depende de uma imagem de sistema e caso não exista nenhuma, pode-se baixar imagens através do "SDK Manager", bastando selecionar um nível de API (neste caso, será utilizada API 27 que é do Android 8.1). Após baixada a imagem, deve-se retornar ao AVD Manager para finalizar a instalação do emulador. Tendo o emulador instalado, restam agora as ferramentas de decompilação android. O APKTOOL pode ser baixado direto pelo aplicativo apt-get em caso de uso de sistemas operacionais Linux baseados na distribuição Debian:

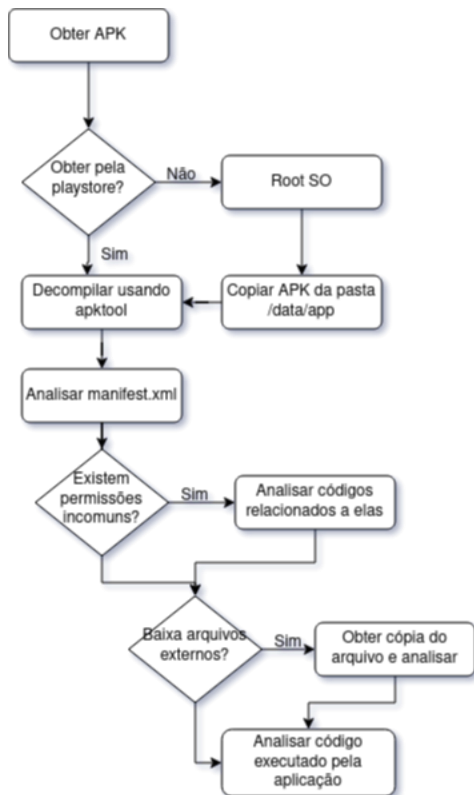
$ sudo apt-get install apktool

Ou então pelo github oficial do projeto `https://github.com/iBotPeaches/Apktool`. Por último, é necessário dirigir-se para as referencias `https://github.com/google/enjarify` e `http://jd.benow.ca` e baixar respectivamente o enjarify e o jd-gui. Com isso, tem-se construído o ambiente controlado para rodar as aplicações e é possível iniciar o processo de análise.

B. Análise estática

A análise estática de programas é um passo fundamental no processo de engenharia reversa com o fim de elucidar como o software age. Esta é definida como o processo de analisar o programa apartir do artefato ou artefatos obtidos lendo-se seu código de acordo com o fluxo de execução e analisando seus dados mas sem executar o programa em algum ambiente. Este último passo é o que caracteriza a análise dinâmica que será abordado mais adiante. Na análise estática foca-se em descobrir o máximo de informações úteis que for possível sobre a aplicação sem que esta seja executada. Muitas vezes esta etapa é mais que suficiente para se chegar a uma conclusão se algum software é malicioso ou não. A seguir veremos um breve fluxograma que descreve de forma ampla e genérica alguns dos principais passos necessários para se investigar um software feito para a plataforma Android.

Figura 1: Fluxograma de análise estática android.



Fonte: De autoria própria

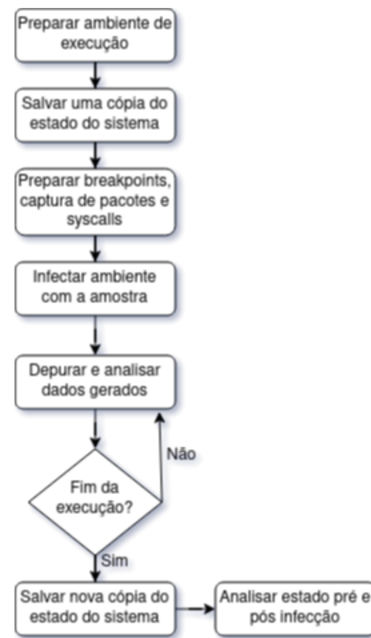
É claro que o exemplo acima é bem abrangente e pode não considerar todos os possíveis cenários encontrados em um ambiente real pois cada programa tem suas características próprias. No entanto, serve de guia para uma grande parte de exemplares que pode se encontrar. Sobre o passo de obtenção do APK, existe um serviço que obtém os instaladores de aplicativos vindos da play store que está disponível no link `https://apps.evozi.com/apk-downloader/` e para se utilizar basta inserir o link de instalação do aplicativo da play store da google que o serviço gera um download do APK para o usuário. Existem casos porém em que isso não é possível mas sempre é possível obter estes arquivos dentro do sistema da pasta /data do Android, mas para isso é necessário que se tenha acesso root ao aparelho. A questão do acesso root e como obter foge do escopo deste texto e também pode ser único para cada aparelho ou marca e por isso não será mostrado aqui.

C. *Análise dinâmica*

Existem muitos casos em que a análise estática pode não ser suficiente para se descobrir todos os passos que o programa malicioso pode executar. Nestes casos é necessário realizar a análise dinâmica da aplicação que se consiste em executar a aplicação em um ambiente controlado, como por exemplo uma sandbox especializada ou máquina virtual, com o fim de observar seu comportamento e verificar quais modificações a aplicação pode fazer no sistema infectado. Tecnicas de ofuscação de código geralmente são muito bem aplicadas resultando em um cenário onde as vezes é mais fácil apenas executar ou debugar para ver com mais exatidão o que a aplicação está fazendo. Durante este tipo de análise é importante levar em conta que o software malicioso pode executar ações danosas no computador ou na rede em que se encontra por isso a importância de serem executados em ambiente controlado.

A análise dinâmica pode mostrar com muito mais detalhes e exatidão o que um programa faz. Durante a execução controlada pode-se, dentre outras ações, coletar dados de rede trafegados entre o sistema para verificar quais informações, a quantidade, a frequência e os destinos delas. Não existe um passo a passo determinístico para a análise dinâmica pelo mesmo motivo da estática mas pode-se definir alguns cenários gerais e comuns conforme o fluxograma a seguir.

Figura 2: Fluxograma de análise dinâmica android.



Fonte: De autoria própria

Cada aplicação maliciosa realiza uma atividade diferente no sistema que ataca, por isso, a condição considerada para "fim de execução"não precisa ser necessariamente ligada ao fim de execução do(s) processo(s) executado(s) pelo malware mas a qualquer momento que o analista do código julgar pertinente. Geralmente todos os malwares alteram o sistema por isso a importancia de salvar uma cópia antes da execução controlada para que seja fácil comparar o que foi modificado ao longo da infecção. O estado do sistema a ser salvo também depende de particularidades de cada caso sendo necessário talvez salvar apenas arquivos de configuração do sistema ou talvez uma cópia inteira de todo o disco e outros artefatos necessários. Mais uma vez, isso dependerá da natureza do programa analisado.

IV. EXPERIMENTOS E ANÁLISE DOS RESULTADOS

As quatro amostras analisadas a seguir foram retiradas do projeto Projeto Android Malware Dataset [8] com acesso efetuado em 2018.

A. *Método para a Avaliação*

O método de avaliação dos exemplares de malwares escolhidos ocorreu de acordo com a metodologia citada acima com ênfase principal na análise estática de malware. Isto foi suficiente para identificar o comportamento do sample, ou seja, apenas lendo seu código embutido sem ter que executá-lo em ambiente controlado. Foi usado principalmente o decompilador dex/java, o apktool e nos casos dos programas que incluiam código nativo embutido, foi utilizado o programa Ghidra para realizar a leitura do código de máquina dos exemplares. Alguns trechos de código de alguns samples não puderam ser traduzidos imediatamente pelo decompilador dex/java. Nestes raros casos a leitura do código foi feita diretamente em smali o que não acarreta em nenhuma forma de perda de informação apesar de tornar a análise um pouco menos amigável.

B. *Análise*

A análise consistiu em manter um registro em planilha onde cada registro é um exemplar identificado pelo hash de seu instalador e as colunas contém as siglas dos traços de comportamentos de malwares conforme ilustrado na tabela 3. Então, foi feita a análise separadamente de cada amostra utilizando as ferramentas já citadas na metodologia. À medida que os exemplares eram lidos e seus comportamentos identificados, tais descobertas eram registradas nas entradas pertinentes.

C. *Demonstração Detalhada de Uma Amostra*

Serão mostradas agora as evidências do sample 1e8d2. Trata-se de uma aplicação que transforma o celular em um bot de envio de SMS. Ao iniciar a análise, no manifest da aplicação também consta o pedido de permissão para executar seu código após cada inicialização para garantir sua persistênia (PE).

Figura 3: Permissões no manifest da aplicação.

```xml
<receiver android:name="com.zaima.SmsReceiver" android:enabled="true">
    <intent-filter android:priority="2147483647">
        <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
        <action android:name="android.intent.action.USER_PRESENT"/>
        <action android:name="android.intent.action.BOOT_COMPLETED"/>
    </intent-filter>
</receiver>
```

Fonte: De autoria própria

No código da aplicação consta a classe SmsReceiver que é uma classe que herda da classe BroadcastReceiver que tem a função de escutar por mensagens difundidas pelo sistema operacional e que são do tipo SMS_RECEIVED, ou seja, para cada SMS recebido.

Continuando, o método handlerSms analisa se a mensagem recebida veio do próprio dono do malware ou de outro lugar e a repassa de volta obtendo o número do telefone a ser enviado a mensagem através da função getSendTo.

Figura 4: Código de envio do SMS.

```java
public static void handlerSms(Context context, String address, String content) {
    String SENDTO2 = getSendTo(context);
    if (address.startsWith("手机号码：")) {
        address = address.substring(3);
    }
    log(context, "收到：" + address + " " + content);
    if (address.equals(SENDTO2)) {
        String[] s = content.split("#", 2);
        if (s.length != 2) {
            log(context, "指令错误");
            sendSms(context, SENDTO2, "短信指令格式错误，格式为“手机号#短信内容”");
            return;
        }
        sendSms(context, s[0], s[1]);
        return;
    }
    sendSms(context, SENDTO2, "[" + address + "]" + content);
}
```

Fonte: De autoria própria

Se o SMS recebido tiver como remetente o próprio número do dono do malware ele age recebendo uma mensagem que contém um novo número de telefone e uma nova mensagem que será repassada pelo SMS do telefone infectado. Ele pode ainda enviar uma mensagem de controle de volta para si caso haja algum erro. Ficando assim caracterizado (IS), (RSM), (SSM), ou seja, roubo de dados, leitura e envio de mensagens SMS.

D. *Avaliação dos Resultados*

Após realizar os passos descritos acima e na metodologia sobre as quatro amostras de malware selecionadas, foi alcançado o seguinte resultado composto pelo tabela a seguir:

Tabela 3: Classificações dos exemplares

Rótulo	82dc7	fcdf2	7940e	1e8d2
RE			X	
TA		X		
TF		X		
ES	X		X	
IP		X		
NB	X	X		
CB	X			
PE	X	X	X	X
IS	X	X	X	X
RSM	X		X	X
SSM			X	X
MB	X			

Fonte: De autoria própria

Foram removidas as linhas características comportamentais que não foram pontuados por nenhum dos exemplares. Os hashs dos quatro arquivos são 43bef74c5d86103d1fd6f3496d182dc7 (ransomware), 3c90cb956a5c3abf8a9728337bbfcdf2
(droid kung fu), 461a6a7ee8b031e351d95f688ef7940e (bankbot) e 992b8f4a45d4350fd5
c8229f6791e8d2 (spambot). Serão utilizados apenas os 5 últimos dígitos de seu hash na identificação assim como na tabela 3. Ficando assim o 82dc7 como ES/NB/CB/PE/IS/RSM/MB, fcdf2 como TA/TF/IP/NB/PE/IS, 7940e como RE/ES
/PE/IS/RSM/SSM e 1e8d2 como PE/IS/RSM/SSM. Note que o nome do último é um subconjunto de seu anterior, ou seja, pode-se entender que o segundo replica os mesmos comportamentos de seu predecessor.

V. CONCLUSÃO

A. *Aplicações práticas e conclusão*

Este trabalho propõe melhorias na taxonomia atual para prever ameaças futuras de programas maliciosos. Em meio ao cenário competitivo entre hackers e analistas de segurança na arena das ameaças digitais, uma metodologia é apresentada para agilizar a análise, identificação e nomeação de malwares, auxiliando os analistas de segurança. A detecção de malwares desconhecidos muitas vezes depende do trabalho humano devido à complexidade das nuances de cada código. Dado que a segurança é constantemente afetada por mudanças e inovações, a identificação de comportamentos bem definidos, como os descritos no trabalho, facilita o reconhecimento em outros casos. Bancos de dados com exemplos e comportamentos categorizados podem auxiliar na detecção futura de malwares desconhecidos, permitindo a criação de funções para analisar e categorizar trechos de código, servindo como base para identificar ameaças futuras por meio de ferramentas automatizadas.

REFERÊNCIAS

[1] OWASP (2023). *Mobile Security Testing Guide*. Acessado em 10 de Agosto de 2023, em: `https://github.com/OWASP/owasp-mstg`.

[2] The Computer Journal (2015). *Toward a taxonomy of malware behaviors*. Grégio, André Ricardo Abed and Afonso, Vitor Monte and Filho, Dario Simões Fernandes and Geus, Paulo Lício de and Jino, Mario.

[3] Tomas Bata University (2013). *Security Reverse Engineering of Mobile Operating Systems: A Summary*. ADEK VALA, LIBOR SARGA, RADEK BENDA.

[4] National Institute of Standards and Technology's National - USA (2019). *National Vulnerability Database*. Endereço: `https://nvd.nist.gov/`.

[5] Auerbach Publications (2014). *Android malware and analysis*. Dunham, Ken and Hartman, Shane and Quintans, Manu and Morales, Jose Andre and Strazzere, Tim.

[6] AOSP (2023). *Android Source*. Acessado em 10 de Agosto de 2023, em: `https://source.android.com/`.

[7] IEEE (2014). *Catch Me If You Can: Evaluating Android Anti-Malware Against Transformation Attacks*. Vaibhav Rastogi, Yan Chen, and Xuxian Jiang.

[8] Springer (2017). *Deep Ground Truth Analysis of Current Android Malware*. Wei, Fengguo and Li, Yuping and Roy, Sankardas and Ou, Xinming and Zhou, Wu. Disponivel em: `http://amd.arguslab.org/`.

[9] Google (2020). *Transform dex into jar files*. Software Enjarify disponivel em `https://github.com/google/enjarify`.

[10] AOSP (2020). *Java Decompiler Project*. Software jd disponivel em `http://jd.benow.ca`.

[11] Google (2023). *Java Decompiler*. Software Java Decompiler disponivel em `http://java-decompiler.github.io/`.

[12] Google (2020). *Java Decompiler*. Linguagem Smali. `https://github.com/JesusFreke/smali`.