Um sistema para análise e detecção de aplicações maliciosas de Android

Vitor M. Afonso[1], Matheus F. de Amorim[1], Eduardo Ellery[1], André R. A. Grégio[1], Glauco B. Junquera[2], Guilherme A. K. Schick[2], Ricardo Dahab[1], Paulo Lício de Geus[1]

[1]Universidade Estadual de Campinas (UNICAMP)

[2]Samsung Instituto de Desenvolvimento para a Informática

Abstract. *The increase in mobile device sales has led to the rise of new malware samples targeting these platforms. This scenario is specially serious for the Android operating system, whose marketplaces (official and alternative) have been used as a point of infection for many users. Therefore, there is a need for the development of techniques to analyze applications from marketplaces and to identify their malicious behavior before publishing them and allowing the users to get infected. In this paper, we present a system to dynamically analyze and to detect (based on machine learning algorithms) malicious Android applications. The tests performed to validate the system were done using thousands of applications, leveraging detection rates of 95,45%.*

Resumo. *O aumento na quantidade de dispositivos móveis vendidos levou ao surgimento de inúmeros exemplares de* malware *para estas plataformas. Essa situação é especialmente grave no caso do sistema Android, cujas lojas (oficial e alternativas) servem como ponto de infecção para muitos usuários. Com isso, faz-se necessário o desenvolvimento de técnicas para analisar aplicações provenientes das lojas de Android e identificar seus comportamentos maliciosos antes que elas sejam obtidas por usuários. Neste trabalho, apresenta-se um ambiente para análise dinâmica e detecção por aprendizado de máquina de* malware *de Android. Os testes para validação do ambiente, realizados com milhares de aplicações, resultaram em uma taxa de detecção de 95,45%.*

1. Introdução

Os dispositivos móveis estão cada vez mais presentes no dia-a-dia das pessoas que, por sua vez, armazenam diversos tipos de informações sensíveis neles, como dados bancários ou documentos confidenciais. Isso torna os ataques contra usuários de dispositivos móveis mais atrativos para os criminosos digitais, consequentemente aumentando a oferta de aplicações maliciosas.

Por ser a plataforma móvel com maior porcentagem de mercado atualmente [Gartner 2012], essa situação é especialmente grave para o Android. Segundo relatório da Juniper Networks [JuniperNetworks 2013], o número de aplicações maliciosas encontradas para todas as plataformas móveis cresceu 614% entre março de 2012 e março de 2013. Além disso, esse mesmo estudo apontou que 92% dos exemplares de *malware* para plataformas móveis têm como alvo a plataforma Android.

As aplicações de Android são disponibilizadas através de lojas. A princípio, o dispositivo obtém apenas aplicações da loja oficial da Google, a Google Play. Entretanto,

pode-se facilmente configurar o dispositivo para obter aplicações de lojas alternativas. Tanto a loja oficial como as lojas alternativas possuem aplicações maliciosas se passando por legítimas na tentativa de enganar os usuários que as obtêm, mesmo que na loja oficial a incidência de aplicações maliciosas seja menor [Zhou et al. 2012].

Antes da disponibilização na loja, as aplicações enviadas ao Google Play são avaliadas pelo Bouncer [Google 2012], que verifica se são maliciosas. Porém, mesmo após o processo de verificação, aplicações maliciosas ainda podem ser encontradas na loja oficial. Além disso, as lojas alternativas podem não ter o mesmo rigor da Google Play ao avaliar as aplicações oferecidas. Assim, são necessárias técnicas para analisar as aplicações e detectar as que contêm comportamentos maliciosos antes que fiquem disponíveis para os usuários e sejam capazes de infectá-los.

Existem diversas técnicas na literatura que tratam desse problema utilizando análise estática ou dinâmica, mas todas sofrem de alguma falha na abrangência ou capacidade de detecção.

Neste trabalho, introduz-se um sistema de análise dinâmica de aplicações Android que tenta suprir as lacunas da área tratando os problemas presentes em outras abordagens. O sistema apresentado é capaz de produzir relatórios automaticamente com informações a respeito do comportamento da aplicação analisada. Durante a análise, são monitoradas as chamadas de funções feitas pela aplicação a APIs (*Application Programming Interfaces)* selecionadas, as execuções de chamadas de sistema e o tráfego de rede.

As informações obtidas das APIs e chamadas de sistema são também utilizadas para classificar a aplicação como maliciosa ou benigna por meio do uso de uma técnica de aprendizado de máquina. Para avaliar a capacidade de detecção da técnica utilizada foram realizados testes com aplicações maliciosas e benignas, cujos resultados mostram que o sistema foi capaz de detectar corretamente 95,78% das aplicações.

A extração de comportamentos de aplicações é uma tarefa importante para analistas de segurança que estão estudando o comportamento de aplicações, seja para gerar relatórios informativos, seja para tratar um incidente envolvendo *malware*. Além disso, dado o rápido surgimento de novas aplicações, é importante que os responsáveis por lojas de aplicações tenham métodos automáticos para verificar se essas aplicações contêm comportamento malicioso antes de as disponibilizarem para o público.

Nesse contexto, o presente artigo visa apresentar um meio mais eficaz para a verificação de aplicações Android. Para isso, pretende-se tornar o sistema acessível ao público, de forma a prover um serviço e ao mesmo tempo obter amostras de aplicações benignas e maliciosas, as quais são importantes no desenvolvimento de trabalhos futuros.

De forma resumida, as principais contribuições deste trabalho são:

- A introdução e detalhamento de um ambiente de análise de aplicações de Android que obtém informações de alto nível pela monitoração de APIs, informações de mais baixo nível pela monitoração de chamadas de sistema e captura informações provenientes do tráfego de rede;
- Uma técnica de detecção de aplicações maliciosas para Android que utiliza aprendizado de máquina para classificação, cujos testes com aplicações maliciosas e benignas resultaram em uma taxa de classificação correta de 95,78%.

O restante deste trabalho está organizado como descrito a seguir. Na Seção 2, apresentam-se os trabalhos relacionados. Na Seção 3, introduz-se o sistema de análise e apresenta-se a técnica de detecção de aplicações maliciosas utilizada. Na Seção 4, são mostrados os testes realizados para avaliar a técnica de detecção aplicada e os resultados obtidos. Por fim, na Seção 5, são apresentadas as conclusões e os trabalhos futuros.

2. Trabalhos Relacionados

Assim como nas plataformas tradicionais, a análise e a detecção de aplicações maliciosas de Android podem ser feitas de maneira estática ou dinâmica. Na análise estática avalia-se a aplicação sem que ela seja executada, utilizando informações como as permissões requisitadas ou o uso de certas APIs consideradas de risco. Já na análise dinâmica, a aplicação é executada dentro de um ambiente controlado e são monitorados diversos tipos de ações executadas pelo exemplar sob análise. A técnica de monitoração varia e pode ser feita dentro ou fora do ambiente de análise. Há ainda abordagens híbridas, as quais se utilizam de uma combinação de análise estática e dinâmica. A seguir, as abordagens acima citadas são apresentadas de acordo com a literatura que trata da análise de dispositivos baseados em Android.

Análise estática. Em [Zhou et al. 2012], os autores apresentam uma técnica para identificação de aplicações maliciosas composta de duas etapas: a primeira utiliza análise estática para identificação de novas amostras de famílias conhecidas e a segunda consiste da aplicação de heurísticas para identificação de exemplares de famílias de *malware* desconhecidas.

Já em [Schmidt et al. 2009], os autores apresentam uma metodologia para detectar aplicações maliciosas de Android que compara chamadas de funções extraídas de aplicações a se analisar com chamadas extraídas de exemplares previamente classificados como *malware*. A extração das chamadas é feita com a ferramenta "readelf".

Para minimizar o trabalho de um analista que precisa verificar manualmente se muitas aplicações são maliciosas, em [Grace et al. 2012] os autores apresentam uma técnica de análise estática baseada em análise de risco. Neste caso, a análise estática é aplicada na identificação tanto de aplicações suspeitas dentro de um conjunto muito grande de aplicações variadas, quanto de trechos de código que levem ao comportamento de risco. O objetivo é auxiliar o analista a tomar uma decisão em relação à maliciosidade ou não de uma aplicação.

Análise dinâmica. Em [Enck et al. 2010], os autores apresentam "TaintDroid", um sistema de monitoramento em tempo real que alerta o usuário quando uma aplicação está tentando evadir informações sensíveis do dispositivo. Para tanto, o sistema aplica uma técnica conhecida como *taint analysis*, a qual é implementada por meio de modificações realizadas no código do sistema operacional Android.

Após a disponibilização do "TaintDroid", o *Honeynet Project* baseou-se nesse sistema para desenvolver o "DroidBox" [DroidBox], um sistema de análise dinâmica de aplicações de Android que monitora violações de privacidade e diversas operações feitas pela aplicação sob análise, como escrita de arquivos, uso de rede e envio de mensagens SMS.

O sistema Andrubis [Andrubis] utiliza ambos os sistemas previamente menciona-

dos, “DroidBox” e “TaintDroid”, a fim de realizar análise dinâmica e prover informações a respeito do comportamento de aplicações. Entretanto, o Andrubis está limitado à API de nível 8 (Android 2.3).

Uma abordagem mais geral é apresentada em [Yan and Yin 2012]. Nesse trabalho, os autores apresentam “DroidScope”, um *framework* que utiliza uma versão instrumentada do emulador de Android para realizar análise dinâmica de aplicações e capturar informações acerca de seu comportamento. Tal sistema permite que sejam acopladas ferramentas para análises mais profundas ou mais gerais, como monitores de chamadas de sistema, de instruções e até de vazamento de informações.

Já com foco na detecção de aplicações maliciosas automaticamente, em [Su et al. 2012] os autores apresentam um *framework* que atua em duas fases. Na primeira é feita uma análise das chamadas de sistema monitoradas com a ferramenta `strace` durante o processo de análise dinâmica em um ambiente controlado e, na segunda fase, verifica-se o tráfego de rede para identificar possíveis falsos-positivos.

Outra abordagem para detecção de comportamentos maliciosos é apresentada em [Burguera et al. 2011]. Neste trabalho, os autores apresentam um *framework* para analisar dinamicamente o comportamento de aplicações para detectar *malware* por meio da coleta de traços de chamadas de sistema de usuários reais, através de *crowdsourcing* e um servidor central. A detecção dos comportamentos maliciosos é feita por meio do agrupamento das informações providas.

Análise híbrida. Em [Spreitzenbarth et al. 2013], os autores apresentam “Mobile-Sandbox”, um sistema que auxilia analistas no entendimento do comportamento de aplicações e que utiliza análise estática e dinâmica. Na etapa estática são obtidas informações sobre as permissões solicitadas e sobre o uso de certas APIs de risco. Para realizar a análise dinâmica, eles utilizam o “DroidBox”, mas, além da monitoração herdada do “DroidBox”, esse sistema monitora chamadas para APIs nativas com a ferramenta `ltrace`. “Mobile-Sandbox” está limitado à análise de aplicações com nível de API menor que 11 (Android 3.0).

Outro sistema que usa tanto análise estática quanto dinâmica é apresentado em [Blasing et al. 2010], sendo este capaz de detectar aplicações suspeitas. A parte estática é responsável por descompilar a aplicação para código Java e então procurar por padrões comuns em aplicações maliciosas, como o uso de `System.getRuntime.exec()` ou funções de reflexão. Já na etapa dinâmica, a aplicação é executada em um ambiente controlado e um componente carregado no *kernel* do Android monitora as chamadas de sistema efetuadas por ela.

3. Descrição do sistema

Nesta seção, o método utilizado para a obtenção do comportamento das aplicações analisadas, a técnica aplicada na detecção de comportamentos maliciosos, as informações extraídas da análise propriamente dita e as limitações da abordagem proposta são descritos.

3.1. Arquitetura do sistema

A obtenção do comportamento das aplicações analisadas é feita por meio de três maneiras distintas: monitoração de chamadas de função presentes em APIs selecionadas;

monitoração das chamadas de sistema efetuadas; monitoração do tráfego de rede produzido durante a execução da aplicação sob análise.

Os dados resultantes da obtenção do comportamento da aplicação são então tratados para se extrair atributos que serão usados na classificação da referida aplicação como maliciosa ou benigna. Em adição, o resultado dessa classificação e as informações extraídas dos traços armazenados são usados para gerar um relatório sobre o comportamento da aplicação analisada. A Figura 1 apresenta uma visão geral da arquitetura do sistema.

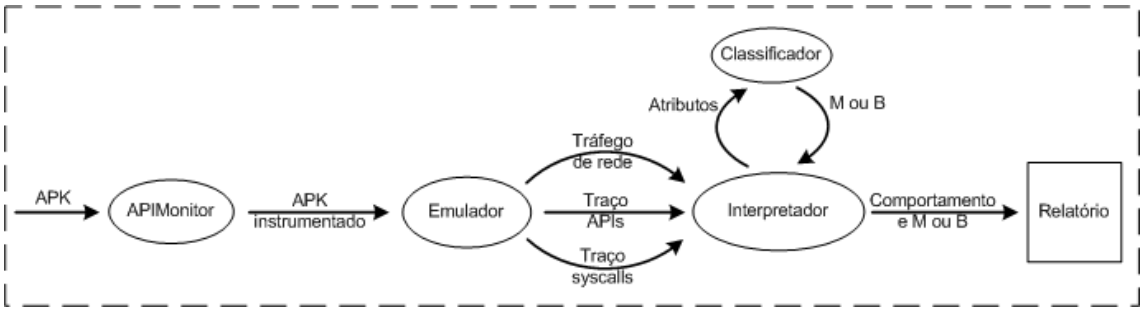

Figura 1. Arquitetura do sistema para análise de aplicações Android.

A aplicação submetida para análise é primeiramente enviada para o "APIMonitor" [APIMonitor] (explicado na Seção 3.1.1), uma ferramenta que gera uma versão instrumentada da aplicação e a envia para o emulador, no qual ela é executada e seu comportamento é monitorado. Após a análise, as informações registradas a respeito das APIs e chamadas de sistema executadas, bem como o tráfego de rede, são passadas para um interpretador, a partir do qual as seguintes etapas são efetuadas: o interpretador extrai os atributos dos dados de API e chamada de sistema e os envia para o classificador; o classificador retorna ao interpretador a informação de que a aplicação é maliciosa ou benigna; o interpretador agrega esses dados a outros dados extraídos dos registros obtidos e gera o relatório de análise, o qual apresenta tanto as informações sobre o comportamento da aplicação, quanto a sua classificação.

O ambiente onde as aplicações são executadas é o emulador de Android padrão, que é distribuído com o Android SDK e é baseado no Qemu [Bellard 2005]. O processo de análise dura cinco minutos e, para tornar o processo de inicialização do ambiente mais rápido, é utilizado um *snapshot* com o ambiente "limpo", ou seja, já iniciado, porém sem nenhuma aplicação instalada além das que estão armazenadas por padrão.

3.1.1. Chamadas a APIs

A monitoração de chamadas a APIs permite a obtenção de informações em alto nível relacionadas ao comportamento da aplicação analisada. Pode-se, por exemplo, obter facilmente o número e texto usados no envio de mensagens SMS ou os números para os quais foram feitas ligações.

Para monitorar as chamadas a APIs, utiliza-se uma versão modificada da ferramenta "APIMonitor" [APIMonitor], desenvolvida pelo *Honeynet Project*. Para o sistema proposto neste artigo, modificou-se a ferramenta de forma a permitir que esta monitore

funções adicionais, as quais foram inseridas no arquivo *default_api*. Esta ferramenta instrumenta a aplicação a ser analisada inserindo códigos que realizam a interceptação do conjunto de APIs selecionadas e o registro dos parâmetros passados a estas funções.

As Listagens 1 e 2 ilustram trechos de exemplo dos registros gerados nessa etapa. A Listagem 1 mostra uma chamada à função de envio de SMS, com número de destino "7132" e mensagem enviada "846978". Já na Listagem 2, é apresentada uma chamada à função de execução de um processo sendo que o processo executado foi `/data/data/org.zenthought.flashrec/cache/asroot` com parâmetros abaixo relacionados:

- `/data/data/org.zenthought.flashrec/cache/explXXXXXX`
- `/data/data/org.zenthought.flashrec/cache/dump_image`
- `recovery`
- `/mnt/sdcard/recovery-backup.img`

Listing 1. Registro do envio de SMS

```
Landroid/telephony/SmsManager;->sendTextMessage(Ljava/lang/String;=7132
| Ljava/lang/String;=null
| Ljava/lang/String;=846978
| Landroid/app/PendingIntent;=null
| Landroid/app/PendingIntent;=null)V
```

Listing 2. Registro da execução de um programa

```
Ljava/lang/Runtime;->exec([Ljava/lang/String;={
/data/data/org.zenthought.flashrec/cache/asroot,
/data/data/org.zenthought.flashrec/cache/explXXXXXX,
/data/data/org.zenthought.flashrec/cache/dump_image,
recovery,
/mnt/sdcard/recovery-backup.img})Ljava/lang/Process;=Process[id=541]
```

As APIs monitoradas pelo sistema são relacionadas a operações de criptografia, rede, manipulação de arquivos, listagem de aplicativos instalados, envio de SMS, realização de chamadas, execução de processos, criação de atividades, criação de serviços, manipulação de *strings*, leitura de informações do dispositivo e operações de banco de dados.

3.1.2. Chamadas de sistema

Existem aplicações que fazem uso de código nativo para executar determinadas atividades, como acessar um recurso do sistema. Nesses casos, a monitoração das APIs com o "APIMonitor" não é capaz de interceptar as ações realizadas. Para contornar esse problema, o sistema apresentado neste trabalho utiliza a monitoração de chamadas de sistema com a ferramenta `strace`. Assim, mesmo quando uma aplicação faz uso de código nativo, o sistema pode monitorar suas atividades, incluindo operações de arquivo e criação de processos.

Para a construção do relatório, são extraídas as seguintes ações encontradas nas chamadas de sistema: processos criados, arquivos acessados, arquivos modificados e conexões de rede realizadas. A Listagem 3 apresenta registros de uma aplicação que acessa

o conteúdo dos arquivos "/proc/cpuinfo" e "/proc/meminfo", obtendo informações relacionadas ao processador e à memória do dispositivo no qual a execução é realizada.

Listing 3. Registros de chamadas de sistema

```
execve("/system/bin/cat", ["/system/bin/cat", "/proc/cpuinfo"],
["ANDROID_SOCKET_zygote=9", "ANDROID_BOOTLOGO=1",
  "EXTERNAL_STORAGE=/mnt/sdcard", "ANDROID_ASSETS=/system/app",
  "PATH=/sbin:/vendor/bin:/system/s"..., 
  "ASEC_MOUNTPOINT=/mnt/asec", "LOOP_MOUNTPOINT=/mnt/obb",
  "BOOTCLASSPATH=/system/framework/"..., "ANDROID_DATA=/data",
  "LD_LIBRARY_PATH=/vendor/lib:/sys"..., "ANDROID_ROOT=/system",
  "ANDROID_PROPERTY_WORKSPACE=8,327"...]) = 0

execve("/system/bin/cat", ["/system/bin/cat", "/proc/meminfo"],
["ANDROID_SOCKET_zygote=9", "ANDROID_BOOTLOGO=1",
  "EXTERNAL_STORAGE=/mnt/sdcard", "ANDROID_ASSETS=/system/app",
  "PATH=/sbin:/vendor/bin:/system/s"..., 
  "ASEC_MOUNTPOINT=/mnt/asec", "LOOP_MOUNTPOINT=/mnt/obb",
  "BOOTCLASSPATH=/system/framework/"..., "ANDROID_DATA=/data",
  "LD_LIBRARY_PATH=/vendor/lib:/sys"..., "ANDROID_ROOT=/system",
  "ANDROID_PROPERTY_WORKSPACE=8,327"...]) = 0
```

3.1.3. Tráfego de rede

A captura do tráfego de rede é feita pelo próprio emulador do Android, sem que a aplicação sob análise possa interferir nesse monitoramento. Atualmente as informações de rede são usadas apenas para prover informações no relatório gerado. Futuramente, tal tráfego poderá ser usado também na etapa de detecção de comportamentos maliciosos e na tentativa de identificar servidores de comando e controle de *botnets* formadas por dispositivos móveis.

Informações a respeito de endereços IP, portas acessadas e requisições de DNS e HTTP são extraídas do tráfego de rede para fins de geração do relatório.

3.1.4. Estímulos

Uma das dificuldades da análise dinâmica é que a monitoração só é capaz de observar atividades provenientes de códigos que forem de fato executados. Assim, é necessário criar estímulos para que mais trechos de código possam ser cobertos e se tenha uma compreensão melhor sobre a aplicação analisada.

No sistema apresentado neste artigo, utiliza-se a ferramenta *MonkeyRunner*, disponibilizada no Android SDK, para gerar eventos para o dispositivo emulado, como toques na tela e pressionamento de teclas. Além disso, como há aplicações que realizam parte das ações apenas quando determinados eventos ocorrem, também é feita a simulação de modificações geográficas e de energia, realização e recebimento de chamadas e envio e recebimento de mensagens SMS.

Por fim, para tornar o ambiente mais parecido com o de um presente no dispositivo móvel de um usuário real, foram alterados o código IMEI e o número de telefone do

sistema, de acordo com os passos informados em [VRT 2013], e foram inseridos contatos para popular a lista do dispositivo emulado.

3.2. Detecção de comportamento malicioso

A detecção de aplicações maliciosas é alcançada por meio da utilização de aprendizado de máquina. Técnicas de aprendizado de máquina permitem que muitas características observadas em exemplares catalogados como maliciosos e benignos sejam utilizadas de forma a gerar um classificador. A geração do classificador tem o objetivo de permitir que se identifique se uma amostra desconhecida é maliciosa ou não.

Atualmente, os atributos utilizados pelo sistema proposto neste artigo são obtidos dos registros gerados pelo "APIMonitor" e pelo `strace`. Os atributos são produzidos a partir da frequência em que cada função monitorada pelo "APIMonitor" foi chamada e a frequência em que cada chamada de sistema foi executada. O processo de treinamento realizado para gerar o classificador é apresentado na Seção 4.2.

A criação do classificador e o processo de classificação são feitos com o framework Weka [Hall et al. 2009], de onde se utilizou o meta-classificador *ThresholdSelection* e o algoritmo classificador *RandomForest*, o qual cria diversas árvores de decisão com seleção aleatória de atributos e então elege a classe que foi selecionada por mais árvores.

3.3. Vantagens e limitações

A obtenção do comportamento de uma aplicação por meio do monitoramento das chamadas a APIs e monitoração de chamadas de sistema com o `strace` envolve apenas a modificação da aplicação a ser analisada e o uso da ferramenta `strace` que já está presente no sistema. Dessa forma, não é necessário modificar o código do Android como alguns sistemas fazem [DroidBox , Spreitzenbarth et al. 2013].

O principal problema de um sistema que necessita de modificações no código do Android é o fato desse código evoluir rapidamente, fazendo com que o sistema tenha que ser constantemente atualizado para a versão mais nova e resultando em uma tarefa árdua e contínua. A monitoração de APIs também passa por alterações quando alguma API é modificada, mas isso ocorre com menor frequência e o trabalho necessário para adequar a portabilidade para a nova API é mais simples, dado que depende apenas da verificação do novo cabeçalho das funções que sofreram alteração.

A captura de informações em alto nível pela monitoração de APIs também facilita a extração de informações relevantes, como mostrado anteriormente. A obtenção de informações de alto nível a partir de dados de baixo nível, como instruções de máquina executadas, é uma tarefa mais complexa.

Por outro lado, as desvantagens do sistema estão relacionadas principalmente a problemas inerentes ao método de análise, ou seja, dinâmico. A análise dinâmica de aplicações é um processo que vem sendo desenvolvido há alguns anos para as plataformas tradicionais (principalmente Windows XP) [Bayer et al. 2006, Willems et al. 2007, Filho et al. 2010]. Entretanto, tais sistemas ainda sofrem de diversos problemas. Por exemplo, se um exemplar de *malware* precisar obter um artefato na Internet ou se conectar a algum servidor de comando e controle para receber instruções do que fazer e essa

conexão não obtiver sucesso, o *malware* pode interromper sua execução, fazendo com que o comportamento suspeito não seja observado. Outro problema que impede que a análise seja feita adequadamente é a detecção desta pela aplicação em execução, seja pela percepção das ferramentas de monitoração ou do ambiente de virtualização/emulação. O sistema apresentado neste trabalho também sofre de tais problemas, como todos os demais que aplicam técnicas de análise dinâmica utilizando ambientes emulados ou virtualizados.

Um quesito no qual o sistema apresentado neste trabalho provê menos informações que outros existentes [DroidBox , Andrubis , Spreitzenbarth et al. 2013] é a falta da análise de vazamento de informações por *taint tracking*. A utilização dessa técnica permite que o sistema verifique o caminho por onde são passadas certas informações sensíveis, gerando um alerta caso sejam enviadas para fora do sistema (por exemplo, pela rede).

Essa deficiência pode ser suprida parcialmente com o uso de assinaturas que identifiquem quando certos dados sensíveis, como o IMEI, o número de telefone ou informações sobre os contatos, estão sendo enviados pela rede ou por SMS. Isso pode ser feito mesmo em casos que utilizam criptografia, porque as operações de criptografia são monitoradas, e em casos nos quais os dados são enviados por HTTPS, porque também são monitoradas as funções que criam objetos usados neste processo antes deles serem criptografados para envio pela rede.

4. Avaliação do método de classificação

Para avaliar a técnica de detecção escolhida neste artigo, foram coletadas 5.476 aplicações, das quais uma parte foi usada no processo de treinamento e outra na etapa de teste. Os conjuntos utilizados incluem exemplares maliciosos e benignos. As subseções a seguir apresentam os conjuntos de dados, o processo de treinamento e o processo de teste.

4.1. Conjuntos de dados

O conjunto de exemplares maliciosos utilizado neste trabalho foi obtido do Projeto Malgenome [Zhou and Jiang 2012] e contém 1.260 aplicações. Para a coleta do conjunto de exemplares benignos, foi desenvolvido um *crawler* utilizado na obtenção das aplicações gratuitas mais populares de cada categoria da loja AndroidPIT [AndroidPIT], que resultou em 4.216 aplicações únicas. Essas aplicações foram submetidas ao VirusTotal [VirusTotal] e aquelas que foram detectadas por pelo menos um de seus antivírus (863 aplicações) foram consideradas suspeitas e retiradas do conjunto de aplicações benignas. Assim, o conjunto resultante de aplicações é dividido em 1.260 exemplares maliciosos e 3.353 exemplares benignos.

O conjunto total de aplicações foi submetido para análise no sistema proposto neste artigo. Algumas aplicações não foram corretamente analisadas, isto é, não produziram todas as informações necessárias ao fim da análise devido a falhas na execução. Tais aplicações foram removidas de seus respectivos conjuntos dada a impossibilidade de serem utilizadas nos processos de treinamento e teste. Assim, o conjunto de aplicações maliciosas ficou com 1.112 exemplares, enquanto que o de aplicações benignas ficou com 3.080. A fim de compor o conjunto de treinamento e gerar o classificador, foram selecionadas aleatoriamente 552 aplicações maliciosas e 1.530 aplicações benignas, formando um total de 2.082 exemplares para treinamento. As 2.110 aplicações restantes formaram o

conjunto de testes para a validação do classificador gerado, composto por 560 aplicações maliciosas e 1.550 benignas.

4.2. Teste de validação

Como mencionado anteriormente, o treinamento para geração do classificador foi feito com 2.082 aplicações, sendo 552 aplicações provenientes do conjunto composto por aplicações maliciosas e 1.530 do conjunto das aplicações benignas. Para isso, utilizou-se a ferramenta Weka, o meta-classificador *ThresholdSelection* e o algoritmo classificador *RandomForest*.

Após a geração do classificador, efetuou-se a validação deste com as 2.110 aplicações do conjunto de testes, sendo 560 aplicações maliciosas e 1.550 aplicações benignas. A taxa de detecção correta foi de 95,45% e a matriz de confusão contendo os resultados deste teste é ilustrada na Tabela 1.

Tabela 1. Matriz de confusão com os resultados do teste para validação do classificador gerado.

		Classe correta		
		Malicioso	**Benigno**	*Total*
Resultado do classificador	**Malicioso**	495	31	*526*
	Benigno	65	1519	*1584*
	Total	560	1550	***2110***

Das 560 aplicações maliciosas, 495 foram corretamente classificadas como tal, enquanto 65 aplicações foram falsos-negativos, isto é, aplicações maliciosas classificadas como benignas. Das 1.550 aplicações benignas, 1.519 foram consideradas desta forma, enquanto 31 foram consideradas maliciosas, formando o conjunto de falsos-positivos.

Existem diversos fatores a se considerar na avaliação dos resultados, como a quantidade de amostras maliciosas que foram detectadas corretamente e a quantidade de amostras benignas que foram classificadas como maliciosas. Por isso, foi utilizada a média harmônica (F-*measure*) na avaliação dos resultados. A média harmônica considera tanto a precisão como o *recall* para calcular a qualidade dos resultados. A precisão indica a fração das amostras classificadas como maliciosas que realmente o são e é calculada por $P = \frac{VP}{(VP+FP)}$, enquanto o *recall* indica a fração das amostras previamente classificadas como maliciosas que foram classificadas assim também pelo sistema sob avaliação e é calculado por $R = \frac{VP}{(VP+FN)}$. Para calcular a média harmônica, é utilizada a fórmula F-*measure* $= \frac{2 \times R \times P}{R+P}$. Os valores de falsos-positivos, falsos-negativos, verdadeiros-positivos, verdadeiros-negativos, precisão, *recall* e média harmônica podem ser encontrados na Tabela 2.

Tabela 2. Valores de falsos-positivos (FP), falsos-negativos (FN), verdadeiros-positivos (VP), verdadeiros-negativos (VN), precisão (P), *recall* (R) e média harmônica (*F-measure*).

FP	FN	VP	VN	P	R	F-measure
2,00%	11,61%	88,39%	98,00%	97,79%	88,39%	92,85%

As amostras classificadas incorretamente estão relacionadas aos falsos-negativos e falsos-positivos. A ocorrência dos falsos-negativos é causada em parte por deficiências na captura dos dados e em parte por limitações dos atributos e algoritmo de classificação utilizados. A captura dos dados pode ser insuficiente para a classificação correta de certos exemplares maliciosos por estes não terem executado as ações maliciosas durante a execução monitorada. Isto pode ocorrer devido a diversos fatores, por exemplo: quando as aplicações esperam por um determinado período de tempo antes de executar as ações maliciosas; porque detectam o ambiente de análise; porque esperam um estímulo específico que não foi feito; por falhas nas ferramentas de monitoração de comportamento. Já os falsos-positivos estão relacionados a limitações dos atributos e algoritmo de classificação utilizados.

O trabalho que mais se assemelha ao método de classificação aplicado neste artigo é o sistema apresentado em [Su et al. 2012], que utiliza chamadas de sistema e tráfego de rede na classificação. Os resultados obtidos em ambos os trabalhos são semelhantes, mas a quantidade de amostras utilizada no teste de validação de [Su et al. 2012] é muito pequeno (50 aplicações maliciosas e 70 benignas), tornando inviável uma comparação mais detalhada da etapa de detecção. Além disso, do ponto de vista das informações que podem ser obtidas, o sistema supracitado não é capaz de obter dados de alto nível como mensagens SMS enviadas e informações enviadas por HTTPS. Por outro lado, o sistema proposto e apresentado neste trabalho é capaz de obter esse tipo de informação, como mencionado nas seções anteriores.

5. Conclusão

Neste trabalho, foi apresentado um sistema de análise dinâmica de aplicações de Android. Tal sistema é capaz de prover informações a respeito do comportamento da aplicação analisada pela monitoração de chamadas de funções de APIs, chamadas de sistema e tráfego de rede. Além disso, o sistema é capaz de classificar a aplicação como maliciosa ou benigna por meio de aprendizado de máquina. O teste de validação da técnica de detecção classificou corretamente 95,45% das aplicações analisadas e teve uma média harmônica de 92,85%.

As principais vantagens do método de análise e monitoração em relação a outros sistemas de análise disponíveis na literatura está na não necessidade de alteração (e consequente recompilação) do código do sistema operacional Android e a obtenção de informações de alto nível sobre o comportamento das aplicações. Essas vantagens, aliadas à qualidade dos resultados produzidos pela técnica de classificação, demonstram a validade e importância do sistema apresentado.

Os trabalhos futuros incluem utilizar, na etapa de detecção, além dos atributos apresentados, atributos obtidos do tráfego de rede e atributos obtidos por análise estática, de forma que seja possível analisar mesmo os casos em que a análise dinâmica tenha falhado. Outra forma de melhorar a detecção é o uso de assinaturas de comportamentos maliciosos, como a execução de determinados processos ou o acesso a URLs contidas em listas de *malware*. Finalmente, de modo a prover mais informações sobre o comportamento da aplicação analisada, uma atividade em andamento é o desenvolvimento de um método de *taint analysis* que detecte o vazamento de informações sensíveis sem que seja necessário alterar o código do sistema Android, como é feito no Taint-

Droid [Enck et al. 2010].

6. Agradecimentos

Parte dos resultados apresentados neste trabalho foi obtida através do Projeto intitulado "Avaliação e prevenção de vulnerabilidades de segurança, em plataformas smartphone e tablet", financiado pela Samsung Eletrônica da Amazônia Ltda., no âmbito da Lei no. 8.248/91.

Referências

AndroidPIT. Androidpit. Disponível em `http://www.androidpit.com.br/`. Acessado em 07 de julho de 2013.

Andrubis. Andrubis: A tool for analyzing unknown android applications. Disponível em `http://anubis.iseclab.org/`. Acessado em 07 de julho de 2013.

APIMonitor. Apimonitor. Disponível em `https://code.google.com/p/droidbox/wiki/APIMonitor`. Acessado em 07 de julho de 2013.

Bayer, U., Kruegel, C., and Kirda, E. (2006). Ttanalyze: A tool for analyzing malware. In *15th European Institute for Computer Antivirus Research (EICAR 2006) Annual Conference*.

Bellard, F. (2005). Qemu, a fast and portable dynamic translator. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '05, pages 41–41, Berkeley, CA, USA. USENIX Association.

Blasing, T., Batyuk, L., Schmidt, A.-D., Camtepe, S. A., and Albayrak, S. (2010). An android application sandbox system for suspicious software detection. In *Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on*, pages 55–62. IEEE.

Burguera, I., Zurutuza, U., and Nadjm-Tehrani, S. (2011). Crowdroid: behavior-based malware detection system for android. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pages 15–26. ACM.

DroidBox. Android application sandbox. Disponível em `https://code.google.com/p/droidbox/`. Acessado em 07 de julho de 2013.

Enck, W., Gilbert, P., Chun, B., Cox, L., Jung, J., McDaniel, P., and Sheth, A. (2010). Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, pages 1–6. USENIX Association.

Filho, D. S. F., Grégio, A. R., Afonso, V. M., DC, R., Santos, M. J., and de Geus, P. L. (2010). Análise comportamental de código malicioso através da monitoração de chamadas de sistema e tráfego de rede. In *Anais do X Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*.

Gartner (2012). Gartner says worldwide sales of mobile phones declined 3 percent in third quarter of 2012; smartphone sales increased 47 percent. Disponível em `http://www.gartner.com/newsroom/id/2237315`. Acessado em 07 de julho de 2013.

Google (2012). Android and security. Disponível em http://googlemobile.blogspot.com.br/2012/02/android-and-security.html. Acessado em 07 de julho de 2013.

Grace, M., Zhou, Y., Zhang, Q., Zou, S., and Jiang, X. (2012). Riskranker: scalable and accurate zero-day android malware detection. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 281–294. ACM.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. (2009). The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18.

JuniperNetworks (2013). Juniper networks mobile threat center third annual mobile threats report: March 2012 through march 2013. Disponível em http://www.juniper.net/us/en/local/pdf/additional-resources/3rd-jnpr-mobile-threats-report-exec-summary.pdf. Acessado em 07 de julho de 2013.

Schmidt, A.-D., Bye, R., Schmidt, H.-G., Clausen, J., Kiraz, O., Yuksel, K. A., Camtepe, S. A., and Albayrak, S. (2009). Static analysis of executables for collaborative malware detection on android. In *Communications, 2009. ICC'09. IEEE International Conference on*, pages 1–5. IEEE.

Spreitzenbarth, M., Freiling, F., Echtler, F., Schreck, T., and Hoffmann, J. (2013). Mobile-sandbox: having a deeper look into android applications. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1808–1815. ACM.

Su, X., Chuah, M., and Tan, G. (2012). Smartphone dual defense protection framework: Detecting malicious applications in android markets. In *Mobile Ad-hoc and Sensor Networks (MSN), 2012 Eighth International Conference on*, pages 153–160. IEEE.

VirusTotal. Virustotal - free online virus, malware and url scanner. Disponível em https://www.virustotal.com/en/. Acessado em 07 de julho de 2013.

VRT (2013). Changing the imei, provider, model, and phone number in the android emulator. Disponível em http://vrt-blog.snort.org/2013/04/changing-imei-provider-model-and-phone.html. Acessado em 07 de julho de 2013.

Willems, C., Holz, T., and Freiling, F. (2007). Toward automated dynamic malware analysis using cwsandbox. *Security & Privacy, IEEE*, 5(2):32–39.

Yan, L. K. and Yin, H. (2012). Droidscope: Seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis. In *Proceedings of the 21st USENIX conference on Security symposium*, pages 29–29. USENIX Association.

Zhou, Y. and Jiang, X. (2012). Dissecting android malware: Characterization and evolution. In *Proceedings of the 33rd IEEE Symposium on Security and Privacy*.

Zhou, Y., Wang, Z., Zhou, W., and Jiang, X. (2012). Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets. In *Proceedings of the 19th Annual Network and Distributed System Security Symposium*.