Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital
IMD0040 - Linguagem De Programação II

Relatório - Insider Threat

Discentes:
Sara Paloma de Souza
Thais Fernandes Lins
Docente:
Carlos Eduardo Da Silva

Natal — RN
2017

Sumário

Projeto de Programação Representando Comportamento de Usuários para Detecção de Ameaças Internas

Sara Paloma de Souza, Thais Fernandes Lins

[1]Instituto Metrópole Digital – Universidade Federal do Rio Grande do Norte (UFRN)

Abstract. *This meta-article describes the implementation and operation of a program destined to analyze the behavior of users for the detection of anomalies and internal threats.*

Resumo. *Este meta-artigo descreve detalhes sobre a implementação e o funcionamento de um programa destinado a analisar o comportamento de usuários para a detecção de anomalias e ameaças internas.*

1. Introdução

Visando uma maior confiabilidade e proteção de acordo com a percepção de estar seguro contra ataques e ameaças no meio virtual, o presente projeto de programação tem como enfoque detecção de ameaças virtuais que podem violar a segurança dos usuários, organizações e empresas que utilizam o meio virtual, e apresentando riscos principalmente as quais armazenam dados ou informações imprescindíveis e sigilosas neste meio.

2. Descrição do Problema Abordado

Em uma época em que ocorre o uso demasiado da internet se faz necessário analisar e ampliar os mecanismos de segurança virtual, já é de conhecimento público alguns exemplos de ataques externos, como o mais recente ataque de ransomware, que ficou popular em maio de 2017, porém, os ataques internos , ou seja, ataques feitos pelos próprios usuários de sistemas, e até usuários com permissões de administradores, embora não seja tão divulgado mas também apresenta muitos riscos a empresas e organizações.

Assim a proposta do programa implementado consiste em estudar o comportamentos desses usuários, o que eles fazem, o que acessam, e tentar detectar um comportamento fora do padrão, ou seja, considerado suspeito, por exemplo, se a maioria dos usuários acessam os mesmos sites, e um usuário está acessando um site muito desconhecido, pode ser considerado um comportamento suspeito.

3. Descrição geral das estruturas de dados utilizadas

As estruturas de dados utilizadas na implementação foram o Map também conhecido como HashMap e o ArrayList. Para guardar as informações de cada perfil de usuário após a leitura de arquivos utilizou-se um HashMap “users” localizado na classe Database, que tem como chave o id do usuário e como valor um objeto do perfil do usuário. Dentro da classe UserProfile (classe do perfil do usuário) há como atributo outro HashMap chamado “devices” que contém as informações dos tipos dos dispositivos que estão relacionados a cada usuário. O HashMap “devices” tem como chave o id do dispositivo e como valor um objeto do tipo PC que corresponde aos dispositivos. Dentro da classe PC utilizou-se um ArrayList para cada tipo de arquivo lido (device.csv, logon.csv,

http.csv) que contém dispositivos e atividades, cada um desses ArrayLists contém objetos do tipo de arquivo respectivo. Essa estrutura de HashMaps e ArrayLists foi utilizada intencionando a criação de uma estrutura hierárquica em que há um usuário como raíz, dispositivos como filhos da raíz, atividades como filhas dos dispositivos e atributos como filhos das atividades, sendo cada nível da estrutura relacionado ao próximo e ao seu nível anterior.

Além disso, na detecção de anomalias a estrutura predominantemente utilizada foi o ArrayList, há um ArrayList como atributo em cada perfil de usuário (classe UserProfile), esse contém 24 posições e será utilizado para realizar a contagem de atividades realizadas pelo usuário em cada hora do dia, a partir dele será realizado uma média de acessos geral de acessos e um desvio padrão, para ambos se utiliza ArrayLists e por fim haverá um ArrayList para armazenar os usuários que são considerados suspeitos ou ameaças.

4. Descrição da abordagem de solução do problema

A solução é baseada no estudo das ações dos usuários, que é passada por um arquivo do tipo csv, após isso é criado um perfil para cada usuário, onde ficam armazenadas as suas informações, e atividades feitas, feito isso o programa verifica todas as atividades de cada usuário por hora e faz uma média por hora. Após isso é feita uma média geral das horas e calculado o desvio padrão de acordo com essas médias.

Finalmente, realiza-se a contagem de acessos realizados por perfil e faz-se uma média desses acessos, se a média do perfil for maior do que a soma do desvio padrão com a média geral o usuário é considerado uma ameaça interna ou um usuário suspeito.

5. Detalhes de projeto OO

Foram criados diagramas de classes de acordo com cada pacote existente no projeto, a fim de se explicar cada processo de implementação detalhadamente e de manter uma estrutura organizada e legível.

5.1. Entrada de Dados

O pacote “filedata” possui as classes referentes às leituras de dados realizadas no sistema, a leitura é o primeiro passo a ser realizado na operação do sistema e portanto foram definidas classes de acordo com cada tipo de arquivo que foi lido, essas são as classes “Device”, “HTTP”, “LDAP” e a classe “Logon”, todas essas classes são filhas de uma classe abstrata “Data” que foi criada para facilitar a criação de objetos de cada tipo, já que cada um pode ser instanciado a partir da classe “Data”. Por fim a classe “FileRead” é a classe que realiza a leitura de cada tipo de arquivo, cria um objeto de acordo com o tipo de arquivo que foi lido e por fim insere os objetos em um HashMap que contém as informações de todos os usuários de acordo com o seu id.

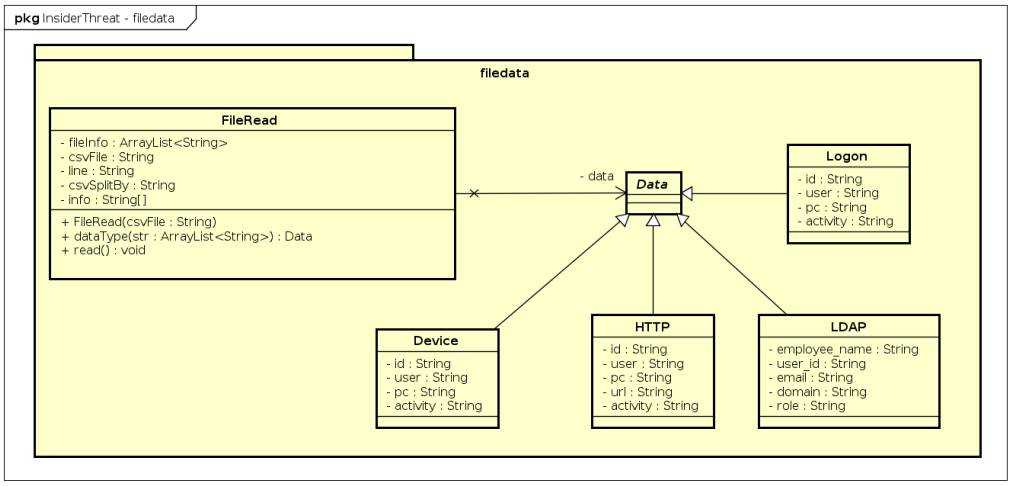

Figura 1. Diagrama de Classes do pacote br.imd.filedata

5.2. Montagem dos Perfis de Usuários

O pacote “profile” possui toda a estrutura de classes de acordo com a montagem do perfil do usuário. A classe “Database” possui somente um HashMap que armazenará todas as informações dos usuários, após as leituras dos arquivos os dados são inseridos no HashMap “users” da classe “Database” por meio dos métodos existentes na classe “ProfileBuilder”. Esse HashMap “users” tem como chave um id de usuário e como valor um objeto da classe “UserProfile” que contém as informações acerca do perfil do usuário, dentro da classe há um atributo “devices” que é um HashMap contendo como chave o id de um dispositivo e como valor um objeto da classe “PC”, que armazena diversos tipos de dispositivos, suas atividades e atributos.

A classe “DateFilter” realiza a filtragem de dados de um usuário existente na database de acordo com um período de tempo especificado. Já a classe “Main” é a classe que realiza a execução e o funcionamento do sistema.

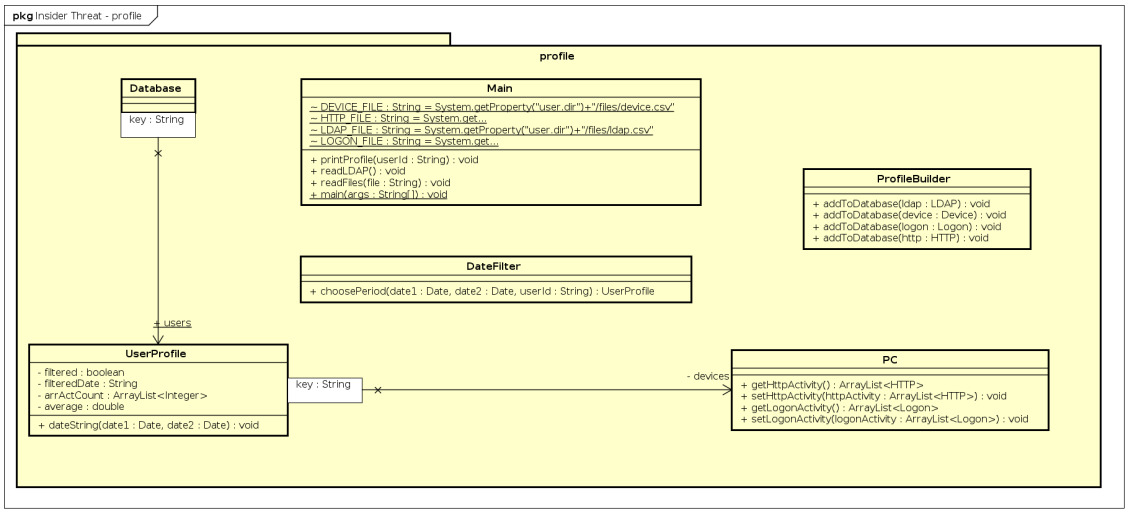

Figura 2. Diagrama de Classes do pacote br.imd.profile

5.3. Visualização dos Perfis de Usuários

O pacote “gui” contém a classe “UserVisualization” que é referente a classe que cria e torna operacional a interface gráfica, nela utilizam-se métodos de diversas outras classes pois certas ações como leitura de arquivos, busca de um usuário que foi inserido e filtragem de dados por período de tempo são ações realizadas em múltiplas classes.

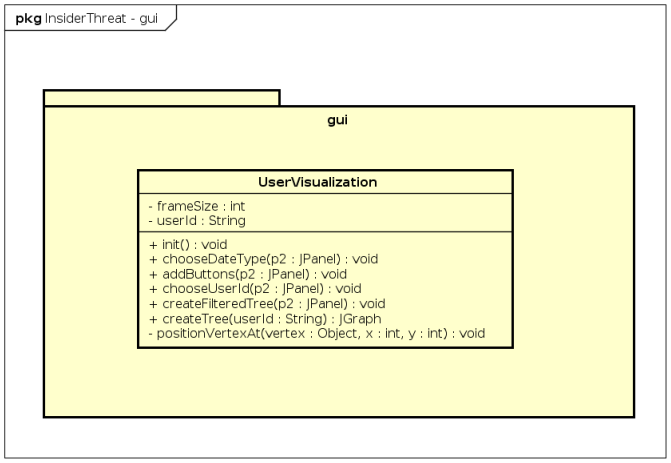

Figura 3. Diagrama de Classes do pacote br.imd.gui

5.4. Detecção de Anomalias

O pacote “anomalies” contém a classe “Threats” que realiza a detecção de anomalias, utiliza-se a database de usuários e os dados já existentes nela e seus perfis a fim de identificar usuários que podem estar realizando atividades suspeitas ou que possam ser considerados ameaças internas.

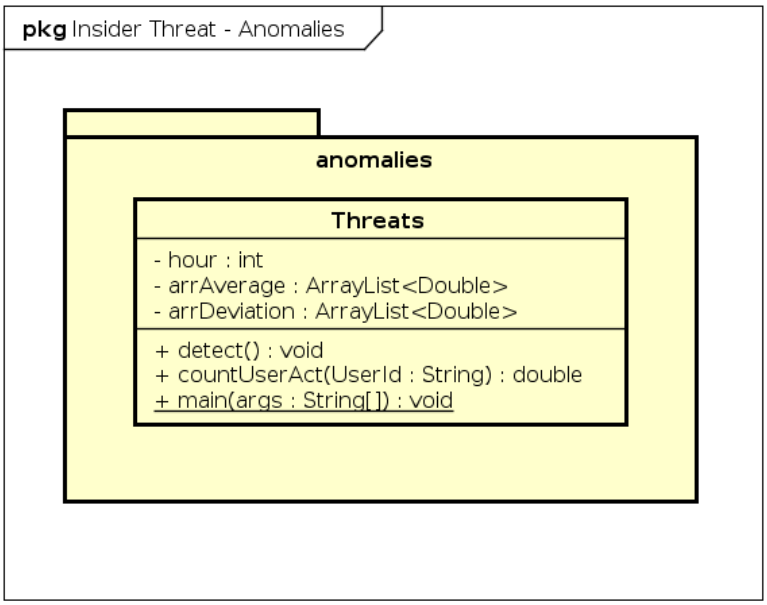

Figura 4. Diagrama de Classes do pacote br.imd.anomalies

5.5. Tratamento de Exceção

O pacote "exception" contém a classe "UserNotFoundException", essa classe possui função de realizar o tratamento de erro na situação em que o usuário procurado não é encontrado na database de usuários.

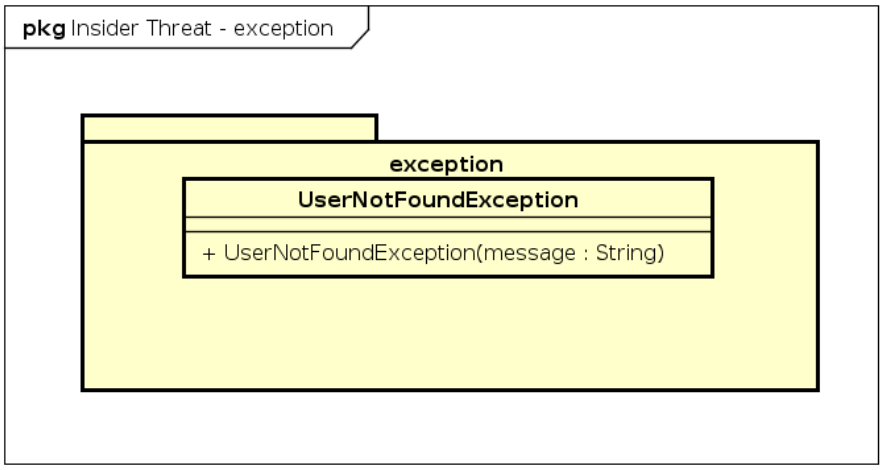

Figura 5. Diagrama de Classes do pacote br.imd.exception

6. Explicação detalhada dos algoritmos utilizados

O algoritmo para leitura dos Logs em formato *CSV* possui complexidade $\Theta(n)$ em relação ao número de linhas do arquivo, tendo em vista que a cada iteração é realizada a conversão para um objeto do tipo *Data* e a inserção no Banco de Dados (*HashMap*), ambos possuindo complexidade $\mathcal{O}(1)$.

Algoritmo para Leitura dos Logs CSV

```
1  input: string logFile
2  begin
3    File file ← readFile(logFile)
4    file.getLine()  // ignora primeira linha do arquivo (header)
5    foreach line in file do
6      fileInfo ← line.split(';')
7      Data data ← parseInfoToDataObject(fileInfo)
8      addDataToDatabase(data)
9    end
10 end
```

O algoritmo para identificação dos usuários suspeitos analisa o comportamento todos os usuários do sistema em relação à quantidade de atividades realizadas diariamente. Com base nessas informações, é calculada a média de acessos de todos os usuários e o desvio padrão por hora, este sendo utilizado para identificar os usuários com comportamento anômalo.

A complexidade desse algoritmo é dada por $\mathcal{O}(n)$ em relação ao número total de atividades no sistema. Tendo em vista que o número de atividades registradas é sempre maior ou igual que o número de usuários cadastrados. Para que todas as atividades possam ser acessadas, é necessário que sejam verificadas as atividades de cada dispositivo de cada usuário. Assim, é necessário o uso de 3 laços aninhados, para os usuários, dispositos e atividades.

Algoritmo para identificação de usuários anômalos

```
input: UserProfile[] users
output: UserProfile[]
begin
  double[24] arrAverage ← fillWith(0.0)
  double[24] arrDeviation ← fillWith(0.0)
  UserProfile[] arrSuspects ← {}
  double totalAverage ← 0.0
  double totalDeviation ← 0.0

  /* Cria array de atividades por hora em cada usuario
     e gera a media de atividades por hora */
  foreach user in users do
    foreach device in user.devices do
      foreach activity in device.activities do
        user.arrActivities[activity.getHour()]++
        arrAverage[activity.getHour()]++
      end
    end
  end

  // Gera a media de atividades por hora
  for i in [0..24] do
    arrAverage[i] ← arrAverage[i]/count(users)
  end

  // Calcula o Desvio Padrao
  foreach user in users do
    for i in [0..24] do
      arrDeviation[i] += pow(user.arrActivities[i] − arrAverage[i], 2)/count(users)
    end
  end
  for i in [0..24] do
    arrDeviation[i] ← sqrt(arrDeviation[i])
  end

  // Identifica usuarios anomalos
  foreach user in users do
    foreach i in [0..24] do
      if user.arrActivities[i] > arrAverage[i] + arrDeviation[i]
         or user.arrActivities[i] < arrAverage[i] − arrDeviation[i] then
        arrSuspects.append(user)
        break  // Go to next user
      end
    end
  end

  return arrSuspects
end
```

7. Conclusão

Assim sendo, o programa é capaz de ler, analisar, mostrar o perfil de um usuário e detectar se ele é anômalo através de comparações com os outros usuários e as atividades realizadas por cada um, tais comparações são feitas através de médias e desvio padrão, o programa embora relativamente simples, é capaz de detectar situações indesejadas em uso de sistemas,sendo uma ferramenta bastante útil e proveitosa.

Referências

[wik 2017] (2017). Segurança da informação.

[Cole 2015] Cole, E. (2015). Insider threats and the need for fast and directed response. SANS Institute InfoSec.

[wik 2017] [Cole 2015]