**Эксперимент**

Для экспериментальной оценки методики был подготовлен стенд со следующими характеристиками:

1) Персональный компьютер intel(R) Core(TM) i9-9880H 2.30GHz, ОЗУ 16 Гб;

2) Операционная система «Linux Ubuntu 20.04»;

3) Дополнительное ПО:

a. Falco security;

b. Docker;

c. Python3;

Для разработки модели машинного обучения на основе “Дерева решений”, необходимы данные событий информационной безопасности “Нормального поведения” и “Аномального поведения” в операционной системе. Для сбора необходимых событий был выбран продукт “Falco security”. Благодаря файлу правил: falco.yaml, мы собираем только те события, которые нам необходимы. Собранные данные попадают в файл events.txt. Также в нашей системе присутствую докер контейнеры, которые в свою очередь имеют уязвимости. С помощью уязвимостей докер контейнеров мы будет выявлять аномальное поведение.

Предполагается, что в системе, для выполнения операции которым необходимы привилегии root, будут выполняться от имени пользователя который будет добавлен в группу sudo, тем самым, от имени самого root операции выполняться не будут.

Falco был сконфигурирован со следующим файлом:

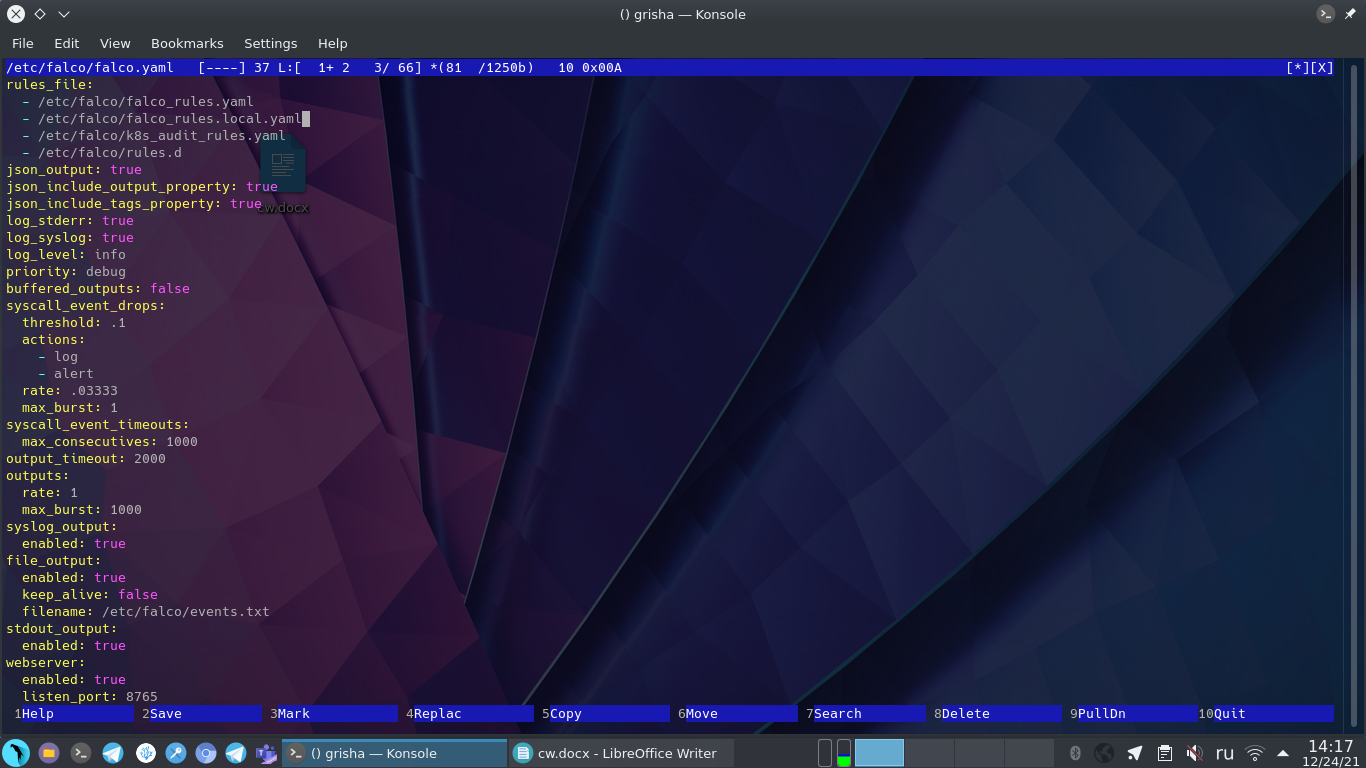
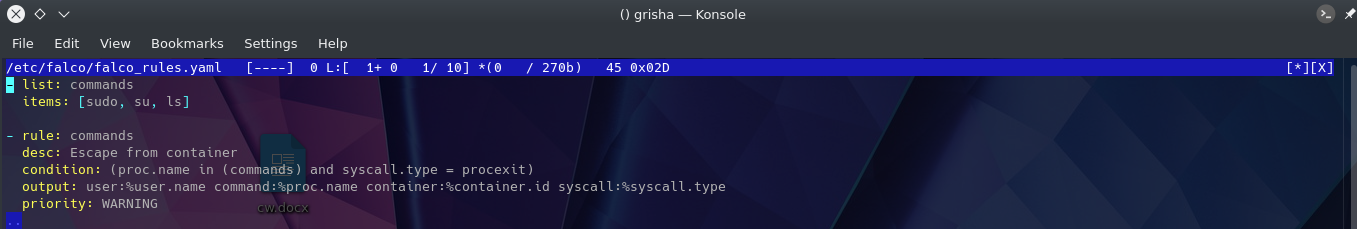


Рис. 2. Файл falco.yaml.

Ссылка на falco.yaml: https://github.com/Uqft/mmi/blob/main/falco/falco.yaml.

Рис. 3. Файл falco\_rules.yaml содержит правила для обнаружения событий.

Ссылка на falco\_rules.yaml: https://github.com/Uqft/mmi/blob/main/falco/falco\_rules.yaml.



Созданы и запущены докер контейнера с использованием docker-compose:

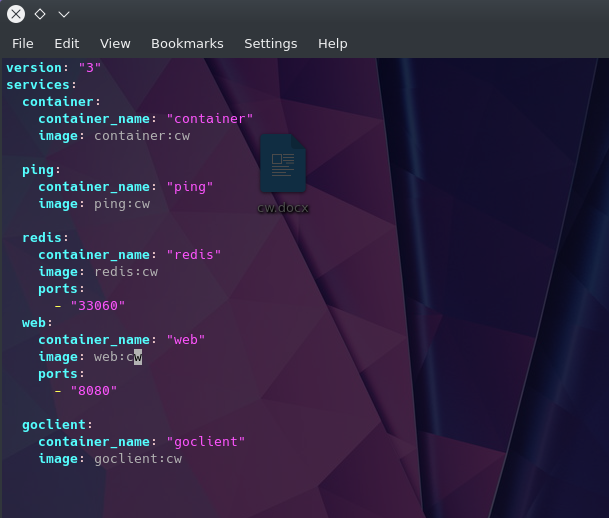


Рис. 4. Файл docker-compose.yml.

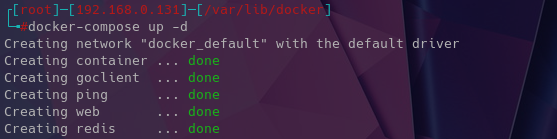


Рис 5. 3апуск докер контейнеров с помощью docker-compose.

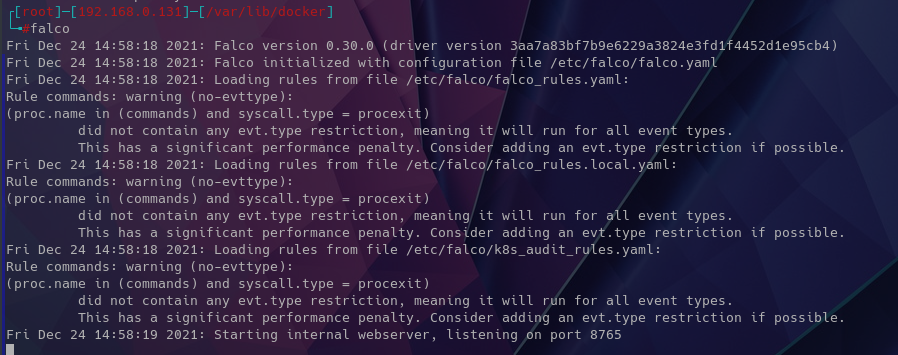


Рис 6. 3апуск Falco.

Falco все события фиксирует в файле events.txt.

Для начала необходимо собрать события информационной безопасности “Нормального поведения”. Результат будет записан в файл: events.txt. Ссылка на файл будет представлена ниже со всеми собранными данными.

Следующим шагом будет предобработка полученных данных, перевод текстов в целочисленное пространство признаков.

Для этой цели был создан скрипт parser\_norm.py на языке python. Скрипт парсит файл events.txt и записывает результат работы в файл table.csv.

Если user\_name root, то в первый столбец записывает значение 1, если нет, то 0.

Если command sudo, то во второй столбец записывает значение 1, если нет, то 0.

Если container\_id host, то в третьий столбец записывает значение 1, если нет, то 0.

В четвертый столбец записывается OK (Символизируя о нормальном поведении).

ok.py:

import csv

import ast

def csv\_create():

header = [["user\_name", "command", "container\_id", "time"]]

logCSV = open('table.csv', 'w')

with logCSV:

writer = csv.writer(logCSV)

writer.writerows(header)

def readlastline(f):

f.seek(-2, 2) # Jump to the second last byte.

while f.read(1) != b"\n": # Until EOL is found ...

f.seek(-2, 1) # ... jump back, over the read byte plus one more.

lastline = f.read()

lastline = lastline.decode("UTF-8")

lastline = ast.literal\_eval(lastline)

return lastline # Read all data from this point on.

def txtlog\_reading():

with open('events.txt',"r") as f:

lines\_list=[]

for line in f:

lines\_list.append(line)

return lines\_list

def main():

message\_list = txtlog\_reading()

print(message\_list[0])

print(type(ast.literal\_eval(message\_list[0])))

for i in range(len(message\_list)):

message = ast.literal\_eval(message\_list[i])

user\_name = message["output\_fields"]["user.name"]

# print(user\_name)

if user\_name == "root":

user\_name = 1

else:

user\_name = 0

command = message["output\_fields"]["proc.name"]

# print(command)

if command == "sudo":

command = 1

else:

command = 0

container\_id = message["output\_fields"]["container.id"]

# print(container\_id)

if container\_id == "host":

container\_id = 1

else:

container\_id = 0

time = message["output\_fields"]["evt.time"]

# print(time)

log = [[user\_name, command, container\_id]]

behavior = "OK"

log = [[user\_name, command, container\_id, behavior]]

logCSV = open('table.csv', 'a')

with logCSV:

writer = csv.writer(logCSV)

writer.writerows(log)

if \_\_name\_\_ == '\_\_main\_\_':

# csv\_create()

main()

Следующим шаго будет исследование “Аномального поведения”. Для этой цели мы будем эксплуатировать уязвимые контейнеры. Контейнеры запущены с root привилегиями.

Это реализация CVE-2019-5736 на Go, побег из контейнера Docker. Эксплойт работает путем перезаписи и выполнения двоичного файла runc хост-системы из контейнера.

Существует два варианта использования эксплойта. Первый вариант является ловушкой. Злоумышленнику потребуется выполнить команду внутри контейнера и запустить вредоносный двоичный файл, который будет его прослушивать. Когда кто-то использует docker exec для проникновения в контейнер, это запускает эксплойт, который разрешает выполнение кода от имени пользователя root.

Exploit и метод атаки был взят отсюда: <https://github.com/Frichetten/CVE-2019-5736-PoC>

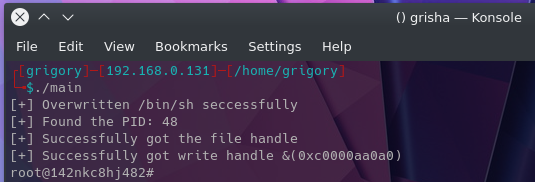


Рис 7. Эксплуатация.

События собираются в файл events.txt. После происходит предобработка и запись в файл table.csv.

Для этой цели был создан скрипт parsernorm.py на языке python. Скрипт парсит файл events.txt и записывает результат работы в файл table.csv.

Если user\_name root, то в первый столбец записывает значение 1, если нет, то 0.

Если command sudo, то во второй столбец записывает значение 1, если нет, то 0.

Если container\_id host, то в третьий столбец записывает значение 1, если нет, то 0.

В четвертый столбец записывается BAD (Символизируя о ненормальном поведении).

bad.py:

import csv

import ast

def csv\_create():

header = [["user\_name", "command", "container\_id", "time"]]

logCSV = open('table.csv', 'w')

with logCSV:

writer = csv.writer(logCSV)

writer.writerows(header)

def readlastline(f):

f.seek(-2, 2) # Jump to the second last byte.

while f.read(1) != b"\n": # Until EOL is found ...

f.seek(-2, 1) # ... jump back, over the read byte plus one more.

lastline = f.read()

lastline = lastline.decode("UTF-8")

lastline = ast.literal\_eval(lastline)

return lastline # Read all data from this point on.

def txtlog\_reading():

with open('events.txt',"r") as f:

lines\_list=[]

for line in f:

lines\_list.append(line)

return lines\_list

def main():

message\_list = txtlog\_reading()

print(message\_list[0])

print(type(ast.literal\_eval(message\_list[0])))

for i in range(len(message\_list)):

message = ast.literal\_eval(message\_list[i])

user\_name = message["output\_fields"]["user.name"]

# print(user\_name)

if user\_name == "root":

user\_name = 1

else:

user\_name = 0

command = message["output\_fields"]["proc.name"]

# print(command)

if command == "sudo":

command = 1

else:

command = 0

container\_id = message["output\_fields"]["container.id"]

# print(container\_id)

if container\_id == "host":

container\_id = 1

else:

container\_id = 0

time = message["output\_fields"]["evt.time"]

# print(time)

log = [[user\_name, command, container\_id]]

behavior = "BAD"

log = [[user\_name, command, container\_id, behavior]]

logCSV = open('table.csv', 'a')

with logCSV:

writer = csv.writer(logCSV)

writer.writerows(log)

if \_\_name\_\_ == '\_\_main\_\_':

# csv\_create()

main()

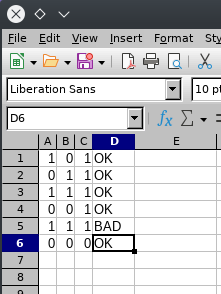


Рис 8. Файл table.csv.

На основе полученных данных мы обучим модель машинного обучения (Дерево принятия решения).

Код модели машинного обучения m.py, метод классификации: Дерево принятия решений. Также мы подаем на вход данные и программа выдает результат.

model.py:

import numpy as np

import pandas as pd

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import accuracy\_score

from sklearn import tree

df = pd.read\_csv('table.csv')

x = df.iloc[:,:3]

y = df.iloc[:,3:]

model=tree.DecisionTreeClassifier(criterion="entropy")

model.fit(x,y)

x, x\_validation, y, y\_validation = train\_test\_split(x, y, test\_size=0.20, random\_state=1)

predictions = model.predict(x\_validation)

print(accuracy\_score(y\_validation, predictions))

print(confusion\_matrix(y\_validation, predictions))

print(classification\_report(y\_validation, predictions))

print (model.predict([[1,1,1]]))

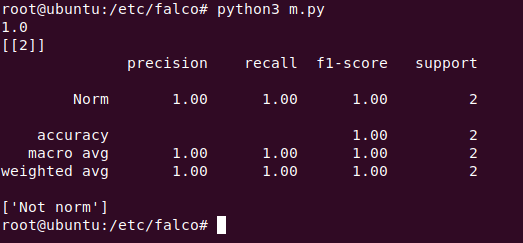


Рис 9. Результат работы.

Также скрипт содержит итоговые значения для классификатора Decision Tree.