Nome: César Rafael Sorgato Santos

Email: rafaelsorgato@hotmail.com

Primeiramente abri o arquivo para analisar manualmente os dados disponíveis, após alguns minutos me familiarizando com o arquivo, verifiquei algumas entradas incomuns no campo do ClientRequestPath, então decidi pegar todos os valores com entradas incomuns através de regex e python:

import pandas as pd

import re

df = pd.read\_excel(r"C:\Users\rafae\Downloads\file.xlsx")

data = []

for index,column in df.iterrows():

    data.append(column["ClientRequestPath"])

itens = []

for item in data:

    if not re.match(r'^/[a-z0-9/]+$', item, re.IGNORECASE):

            if item not in itens:  
 itens.append(item)

df\_itens = pd.DataFrame(itens, columns=['ClientRequestPath'])

with pd.ExcelWriter(r"C:\Users\rafae\Downloads\file.xlsx", mode='a', if\_sheet\_exists='replace') as writer:

    df\_itens.to\_excel(writer, sheet\_name='clientpath', index=False)

Foram retornadas 29 linhas que representam resulados resultados importantes que disponibilizam diversas informações relevantes:

/<img src='x' onerror='alert(1)'>

/admin.php?user=admin&password=admin

/reset-password?email=%27%20OR%201=1--

/cgi-bin/test-cgi

/../../../../windows/system32/cmd.exe

/favicon.ico

/../../../windows/win.ini

/shell.php?cmd=cat%20/etc/passwd

/<script>alert('XSS')</script>

/vulnerable.php?search=<svg onload=alert(1)>

/index.php?id=' OR '1'='1

/../../../../../../../../../../etc/shadow

/script.php?=<script>window.location='http://evil.com'</script>

/<iframe src='javascript:alert(1)'></iframe>

/"!--"<XSS>=&{()}

/.git/config

/config.php.bak

/login.jsp?user=admin'--

/<marquee><img src=1 onerror=alert(1)></marquee>

/wp-admin/setup-config.php

/robots.txt

/../../boot.ini

/%00%01%02%03%04%05%06%07

/../../../etc/passwd

/eval?param=<%=Runtime.getRuntime().exec("calc.exe")%>

/<meta http-equiv='refresh' content='0url=javascript:alert(1)'>

/search?q=<script>alert(0)</script>

/phpinfo.php

/api/v1/users?search= DROP TABLE users--

## Risco 1 - Injeção

Alguns exemplos encontrados:  
/<script>alert('XSS')</script> (Cross site scripting)

/<marquee><img src=1 onerror=alert(1)></marquee> (Cross site scripting)

/api/v1/users?search= DROP TABLE users-- (SQL injection)

**Risco**: Usuários mal intencionados podem executar scripts/querys maliciosas e prejudicar a segurança através de acessos indevidos, perda de dados, comprometimento na integridade dos dados e de sua disponibilidade.  
  
**Politica** -   
  
**Objetivo**: Bloquear e filtrar entradas potencialmente maliciosas sendo requisitadas pela web.

**Escopo**: Todos os sistemas que disponibilizam aplicações web e APIs.

**Responsabilidades**:  
- Gestão - Alocar recursos e garantir o cumprimento.

- Desenvolvimento - Implementar os recursos necessários.

- Segurança da informação - Responder a incidentes, monitorar e testar os sistemas de filtragem e bloqueio.

- Usuários - Reportar qualquer suspeita de vulnerabilidade ou atividade maliciosa.

**Controles**:

- Implementação de sistemas de filtragem.

- Revisão de código.

- Testes rotineiros da eficácia proteção.

- Ajustes de configurações.

**Implementação -**

Sistema de filtragem: Há inúmeras maneiras para efetuar o bloqueio de injeção (WAF, TAGs HTML, código, etc). Pensei em treinar um modelo com um [dataset](https://raw.githubusercontent.com/fmereani/Cross-Site-Scripting-XSS/refs/heads/master/XSSDataSets/Payloads.csv) de XSS para realizar a detecção de inputs maliciosos (lembrando que foi apenas treinado para XSS, há outras entradas que podem ser treinadas, porém pelo tempo disponível, deixei apenas XSS).

import pandas as pd  
from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.naive\_bayes import MultinomialNB

import chardet

with open(r'C:\Users\rafae\Downloads\tools\scripts/xss.csv', 'rb') as f:

    result = chardet.detect(f.read())

df = pd.read\_csv(r'C:\Users\rafae\Downloads\tools\scripts/xss.csv', encoding=result['encoding'])

X = df['Payloads']

y = df['Class']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

vectorizer = TfidfVectorizer(stop\_words='english', max\_features=5000)

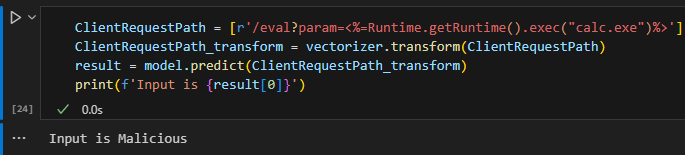
X\_train\_tfidf = vectorizer.fit\_transform(X\_train)

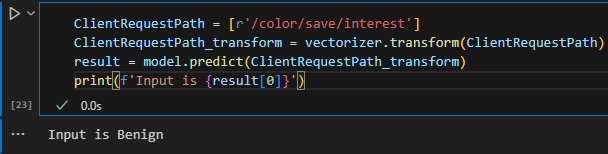
X\_test\_tfidf = vectorizer.transform(X\_test)

model = MultinomialNB()

model.fit(X\_train\_tfidf, y\_train)

Após treinada, seria necessário apenas sua chamada, passando como parâmetro o ClientRequestPath.





Caso sejam quantidades muito grandes de requisições, o código pode ser melhorado em performance adicionando uma validação anterior a predição, como por exemplo a validação de caracteres especiais como “\*!@$<>#%”, na ausência desses ou outros símbolos, não é necessário realizar a predição.

**Testes rotineiros**: Há diversas soluções capazes de realizar testes de entradas maliciosas, um exemplo é o XSStrike que automatiza testes de XSS, dessa forma necessitamos apenas baixar o software e aplicar os testes necessários, não será entrado em detalhes pois não faz parte do desafio, mas fica de bônus:



## Risco 2 - Risco de má configurações

Alguns exemplos encontrados:  
/api/v1/users (Versionamento de API disponível para uso, permitir apenas uso da versão atual)

/wp-admin/setup-config.php (Exposição de Arquivo)

/../../../windows/win.ini (Ataque de Diretório Transversal)

/config.php.bak (Exposição de Arquivo)

**Risco**: Usuários mal intencionados podem prejudicar a segurança através de acessos indevidos, perda de dados, comprometimento na integridade dos dados e de sua disponibilidade.  
  
**Politica** -   
  
**Objetivo**: Revisar, manter e ajustar configurações de aplicações/servidores web, de forma a garantir que estão seguras.

**Escopo**: Todos os sistemas que disponibilizam aplicações web e APIs.

**Responsabilidades**:  
- Gestão - Alocar recursos e garantir o cumprimento.

- Desenvolvimento - Implementar os recursos necessários e configurar os sistemas.

- Segurança da informação - Responder a incidentes, monitorar e testar as configurações dos sistemas web.

- Usuários - Reportar qualquer suspeita de vulnerabilidade ou atividade maliciosa.

**Controles**:

- Implementação de sistemas de filtragem.

- Revisão de código.

- Testes rotineiros da eficácia das proteções.

- Ajustes de configurações.

**Implementação -**

1 - Configurar manualmente as aplicações de acordo com a necessidade e o uso, por exemplo:  
Para o caminho que foi acessado “/cgi-bin/test-cgi”, é necessário saber qual servidor está hospedando (apache por exemplo) e entender se o CGI está sendo utilizado, para restringir o acesso ou desabilitar em caso de não ser utilizado. O mesmo vale para outros casos do tipo, como por exemplo: /config.php.bak e /wp-admin/setup-config.php.

2 - Detectar (Ataque de Diretório Transversal)

O ideal para esse passo é corrigir má configurações, porém se for necessário realizar através de scripts, é possível utilizar diversas [wordlists](https://github.com/Karanxa/Bug-Bounty-Wordlists/tree/main) em conjunto para detectar padrões já mapeados de má configurações (o exemplo abaixo vai incluir tanto exemplos de ataque de diretório transversal quanto os casos de má configurações do passo 1):

Nesse exemplo serão usadas duas listas ([config.txt](https://github.com/Karanxa/Bug-Bounty-Wordlists/blob/main/config.tx) e [php\_files\_with\_path.txt](https://github.com/Karanxa/Bug-Bounty-Wordlists/blob/main/php_files_with_path.txt)), porém essas listas vão depender das aplicações utilizadas no ambiente.

O código valida os 2 tipos de requisições que podem chegar (as encodadas e as não encodadas)

import urllib.parse

import pandas as pd

miss\_patterns = [line.strip() for line in open(r'C:\Users\rafae\Downloads\tools\scripts/leaky-misconfigs.txt')]

def Filter(ClientRequestPath):

    if any(character in ClientRequestPath for character in["%3B" , "%3C" , r"%3E" , "%0A" , "%0D" , "%20" , "%00"]):

        return "BAD REQUEST"

    else:

        ClientRequestPath = urllib.parse.unquote(ClientRequestPath)

        if("../") in ClientRequestPath:

            return "BAD REQUEST"

        elif any(pattern in ClientRequestPath for pattern in miss\_patterns):

            return "BAD REQUEST"

        else:

            return "PASS"

print(Filter('/../../../windows/win.ini'))

print(Filter(f'%2F..%2F..%2F..%2Fwindows%2Fwin.ini'))

print(Filter('/color/save/interest'))

print(Filter('/wp-admin/setup-config.php'))

print(Filter('/.git/config'))

**Resultados**:

BAD REQUEST

BAD REQUEST

PASS

BAD REQUEST

BAD REQUEST

## Risco 3 - Suspect user-agents

Alguns exemplos encontrados:  
Mozilla/5.0 (compatible MSIE 5.0 Windows 98 Win 9x 4.90 Trident/5.1)  
Mozilla/5.0 (Android 1.0 Mobile rv:56.0) Gecko/56.0 Firefox/56.0

Mozilla/5.0 (iPod U CPU iPhone OS 3\_2 like Mac OS X eu-FR) AppleWebKit/532.20.4 (KHTML, like Gecko) Version/3.0.5 Mobile/8B116 Safari/6532.20.4

**Risco**: Usuários mal intencionados estão alterando dados de user-agent, podendo estar sendo utilizado para automações, como o bruteforce.  
  
**Politica** -   
**Objetivo**: Bloquear user-agents supeitos.

**Escopo**: Todos os sistemas que disponibilizam aplicações web e APIs.

**Responsabilidades**:  
- Gestão - Alocar recursos e garantir o cumprimento.

- Desenvolvimento - Implementar os recursos necessários.

- Segurança da informação - Responder a incidentes, monitorar e testar os sistemas de bloqueio.

- Usuários - Reportar qualquer suspeita de vulnerabilidade ou atividade maliciosa.

**Controles**:

- Implementação de sistemas de bloqueio.

- Testes rotineiros da eficácia proteção.

**Implementação -**

Sistema de bloqueio

Em relação ao user-agent, é uma tarefa complicada sair bloqueando pois pode afetar usuários comuns, então vai depender da regra de negócio, no caso o script abaixo é focado somente para user-agents extremamente suspeitos, pois contem versões extremamente antigas que não circulam mais no mercado de forma comum, além de não possuírem compatibilidade com páginas web da atualidade (Praticamente todos user-agents do documento parecem ter sido gerados manualmente, mas estarei ignorando isso e seguindo no pensamento de grande parte ser confiável). No código abaixo considerei versões defasadas de alguns sistemas operacionais para considerar como suspeitos, suas buscas são feitas através de regex.

import re

def Filter(ClientRequestUserAgent):

    if re.match(r'.+(Android [1-3]\.|Windows (9(8|5)|NT)|iPad OS [1-5]\_|iPod)', ClientRequestUserAgent):

        return "SUSPICIOUS"

    else:

        return "Not suspicious"

print(Filter("Mozilla/5.0 (iPad CPU iPad OS 1\_1\_5 like Mac OS X) AppleWebKit/536.1 (KHTML, like Gecko) CriOS/60.0.896.0 Mobile/31N563 Safari/536.1"))

print(Filter("Mozilla/5.0 (Android 12.1 Mobile rv:38.0) Gecko/38.0 Firefox/38.0"))

print(Filter("Mozilla/5.0 (Android 1.1 Mobile rv:33.0) Gecko/33.0 Firefox/33.0"))

print(Filter("Opera/9.99.(Windows NT 5.0 or-IN) Presto/2.9.172 Version/10.00"))

RETORNOS:  
SUSPICIOUS

Not suspicious

SUSPICIOUS

SUSPICIOUS

## Risco 4 - Falta de SSL/TLS

Alguns exemplos encontrados:  
thomas.biz - http

pena.com - http

sosa.com - http

valdez-clarke.com - http

marshall-odonnell.net - http

**Risco**: Usuários mal intencionados podem interceptar dados descriptografados que transitam pela rede.  
  
**Politica** -   
**Objetivo**: Implementar SSL/TLS.

**Escopo**: Todos os sistemas que disponibilizam aplicações web e APIs.

**Responsabilidades**:  
- Gestão - Alocar recursos e garantir o cumprimento.

- Desenvolvimento - Implementar os recursos necessários.

- Segurança da informação - Responder a incidentes e monitorar os certificados.

- Usuários - Reportar qualquer certificado expirado.

**Controles**:

- Implementação do SSL/TLS.

- Monitoramento.

- Sistema de bloqueio

**Implementação -**

Sistema de bloqueio

Referente ao bloqueio de requisições http vai depender da regra de negócio, há empresas que optam por não usar criptografia por redução de custos ou por não necessitar de criptografia devido a natureza da página.

O código abaixo seta quais métodos HTTP não são permitidos em requisições “http”

BlockedMethods = ["GET","POST"]

def Filter(ClientRequestScheme, Method):

    if 'http' == ClientRequestScheme:

        if Method in BlockedMethods:

            return "HTTP BLOCKED"

return "PASS"

print(Filter("http","GET"))

print(Filter("http","POST"))

print(Filter("http","PATCH"))

print(Filter("https","POST"))

RETORNO:  
HTTP BLOCKED

HTTP BLOCKED

PASS

PASS

Há outras políticas que poderiam ser implementados, como bloqueio de ClientDeviceType = Desktop enquanto o user-agent aponta para um mobile; Bloqueio de acessos com IP em país X enquanto o ClientCountry aponta para país Y (113.50.114.50 = CN | ClientCountry = IN), o mesmo vale para validação do País entre o IP/ClientCountry com a ASN, visto que possuem regiões fixas.  
Porém, esgotei meu tempo, então estarei enviando com os dados que tive tempo.