Cookies Extractor Tool & the Dark Web

Fotoon Samman – Ameerah Alturki – Danah Alrehaili – Lama Alhazmi

King Abdullah University of Science and Technology Academy

Cyber Security Summer School

supervisor name

August 20, 2024

Abstract

This report aims to document the process of building a tool that extracts, analyzes, and scans the dark web for compromised cookies.

Keywords: Cookies, Cyber forensics, Cybersecurity, Dark web, Web forensics

In the hidden corners of the dark web, stolen cookies are traded like digital assets, granting unauthorized access to online services and violating user privacy, posing a growing threat to personal security. Current forensic tools lack in-depth cookie analysis and dark web integration, leading to incomplete investigations and prolonged exposure to security threats. The proposed solution is a forensic cookie extractor tool that not only extracts and analyzes cookies from various web browsers but also integrates with dark web monitoring services. This integration allows the tool to cross-reference extracted cookies with known dark web databases, providing real-time alerts if any cookies have been compromised and are being traded illicitly. This report aims to document the process of building a tool that extracts, analyzes, and scans the dark web for compromised cookies. The following sections will cover the methodology and results of the cookies extractor tool.

Methodology

From conception and design to execution and testing, there were multiple phases involved in the development of the forensic cookie extractor technology. The steps that represent the approach used in this report are as follows:

1- Tool Design and Requirement Analysis: The first stage of the project required determining the essential functional needs for the tool, such as the ability to extract cookies from different web

browsers, carry out in-depth cookie analyses, and integrate with services that monitor the dark web. The design prioritized compatibility with popular online browsers, and smooth integration with already-existing dark web databases.

2- The development of the cookie extraction tool: This stage involved building the cookie extraction functionality for popular web browsers, The tool was made to function in a non-intrusive manner so that cookie extraction may be done without interfering with the user's browsing experience.

3- Cookie Analysis and Classification: Following the extraction process, the tool performed detailed analysis of the retrieved cookies. This involved classifying cookies according to their type (e.g., necessary, functional cookies) and purpose (e.g., tracking, session cookies).

4- Integration with Dark Web Monitoring Services: One of the tool's most important features was its integration with these services. The program was integrated with dark web monitoring services and databases through the use of APIs. The tool cross-reference the retrieved cookies with known compromised data in real-time by scanning these platforms. The detection of stolen cookies being traded or sold illegally was made possible by this integration.

5- Real-Time Alerts: In order to improve security, users were notified if any of their retrieved cookies were found on the dark web. The system's ability to alert users enables them to take immediate action to secure their accounts by updating their passwords or taking other required steps.

Results

Our results were as follows:

```
┌──(lama㉿kali)-[~/forensic_cookie_extractor]
└─$ python CookieGuard.py --browser firefox --output firefox_cookies6.json
```

The above command demonstrates the arguments here are the browser ”Firefox” and output file “firefox_cookies6.json”

~/forensic_cookie_extractor/firefox_cookies2.json - Mousepad

File Edit Search View Document Help

```
1296    },
1297    {
1298        "Domain": "www.exploit-db.com",
1299        "Cookie Name": "XSRF-TOKEN",
1300        "Cookie Value":
1301        "Type": "Functional Cookie",
1302        "Purpose": "To authenticate users and maintain session state."
1303    },
1304    {
1305        "Domain": "www.exploit-db.com",
1306        "Cookie Name": "exploit_database_session",
1307        "Cookie Value":
1308        "Type": "Necessary Cookie",
1309        "Purpose": "To manage user sessions and maintain state across pages."
1310    },
1311    {
1312        "Domain": "www.exploit-db.com",
1313        "Cookie Name": "CookieConsent",
1314        "Cookie Value":
1315        "Type": "Functional Cookie",
1316        "Purpose": "To store the user's consent preferences regarding cookies and privacy."
1317    },
```

Above are the analysis results. it is based on a user criteria or alternatively a keyword search or string search.

```python
def format_cookie_output(host, name, value):
    cookie_type, purpose = analyze_cookie(name)
    return {
        "Domain": host,
        "Cookie Name": name,
        "Cookie Value": value,
        "Type": cookie_type,
        "Purpose": purpose
    }

def extract_chrome_cookies(output_file):
    path = os.path.expanduser('~/.config/google-chrome/Default/Cookies')
    conn = sqlite3.connect(path)
    cursor = conn.cursor()
    cursor.execute("SELECT host_key, name, value, encrypted_value FROM cookies")

    cookies = []
    for row in cursor.fetchall():
        host, name, value, enc_value = row
        cookie_value = value or enc_value
        formatted_output = format_cookie_output(host, name, cookie_value)
        cookies.append(formatted_output)

    with open(output_file, 'w') as f:
        json.dump(cookies, f, indent=4)

    conn.close()

def extract_firefox_cookies(output_file):
    path = os.path.expanduser('~/.mozilla/firefox/03fo503o.default-esr/cookies.sqlite')
    conn = sqlite3.connect(path)
    cursor = conn.cursor()
    cursor.execute("SELECT host, name, value FROM moz_cookies")

    cookies = []
    for row in cursor.fetchall():
        host, name, value = row
        formatted_output = format_cookie_output(host, name, value)
```

The above code will format the output file in whatever way the user wants; exactly which attributes will be included. For us, we chose cookie names and values and domain and type and purpose. The type and purpose will be selected based on the previous analysis. it also extracts cookies from Chrome and Firefox.

```python
# API - Twingly

def parse_cookies_from_json(json_file_path):
    with open(json_file_path, 'r') as file:
        cookies = json.load(file)
    return cookies

def alert_if_match_found(api_response):
    if api_response.get("number_of_matches_total", 0) > 0:
        print("Alert: One or more matches were found for your cookies in the dark web.")
    else:
        print("No matches found for your cookies in the dark web.")

def check_cookies(json_file_path, response_file_path='api_response.json'):
    cookies = parse_cookies_from_json(json_file_path)

    url = 'https://data.twingly.net/darkweb/a/search/v1/search'
    headers = {
        'Authorization': '[redacted]',
        'Content-Type': 'application/json; charset=utf-8',
        'Accept': 'application/json; charset=utf-8'
    }

    queries = []
    for cookie in cookies:
        name = cookie["Cookie Name"]
        value = cookie["Cookie Value"]
        queries.append(f'cookie_name:"{name}" AND cookie_value:"{value}"')

    query_string = " OR ".join(queries)
    payload = {
        'query': query_string[:2000]  # Truncate to 2000 characters
    }
```

Then the first function, parse_cookies_from_json, will read the information from the json file which contains the cookies we previously extracted. The next function, alert_if_match_was_found, is an API response which is a file containing the total number of matches found, if it is less than 0 it will inform the user that no matches were found, but if it was more than 0 it will alert the user. The next function, check_cookies, is the function connecting us with the API, it takes two arguments: the json file and the response file. we blocked the Authorization as it is our API key. it currently only accepts json files. The queries are specifying what we are looking for in the dark web. There is an API limit of 2000 characters.

```
116     response = requests.post(url, json=payload, headers=headers)
117
118     if response.status_code == 200:
119         response_json = response.json()
120         with open(response_file_path, 'w') as response_file:
121             json.dump(response_json, response_file, indent=4)
122
123         # Check if any matches were found and alert the user
124         alert_if_match_found(response_json)
125
126         return response_file_path
127     else:
128         error_response = {"error": "Failed to connect to the API"}
129         with open(response_file_path, 'w') as response_file:
130             json.dump(error_response, response_file, indent=4)
131         return response_file_path
132
133
134 if __name__ == "__main__":
135     args = parse_args()
136     if args.browser.lower() == "chrome":
137         extract_chrome_cookies(args.output)
138     elif args.browser.lower() == "firefox":
139         extract_firefox_cookies(args.output)
140     else:
141         print("Unsupported browser")
142
143     # Check cookies with the Twingly API
144     response_file = check_cookies(args.output, 'api_response.json')
145     print(f"The API response has been saved to: {response_file}")
146
```

Above is the rest of the code. if there was an error, we have an error response.

```
┌──(lama㉿kali)-[~/forensic_cookie_extractor]
└─$ python CookieGuard.py --browser firefox --output firefox_cookies8.json
No matches found for your cookies in the dark web.
The API response has been saved to: api_response.json
```

Above is the response. Thankfully, no cookies of ours were found on the dark web!



```
{
    "number_of_matches_total": 0,
    "number_of_matches_returned": 0,
    "documents": [],
    "warnings": [
        {
            "message": "Unknown warning",
            "severity": "error"
        },
        {
            "message": "Unknown warning",
            "severity": "error"
        }
    ],
    "next_cursor":
"e2cacf9f57443cc5b5a2bcc5bed2ed7474b41eb1257d866824f77d23d7102cf832c6027572cd987a35d5d4247ac7cd1c332481c128505408
2d7ef479384a6774d2129a3fd8ec6be18538ba5e35f93855864239bd00158ad0bfb2f99780566aca97d2793234f3c6d29826e170a9d6181cc
dc38e4983bccf6881e0ebcaf1c5eb6d5f254b74d60045f1c56181b3d83663f56cdd6a16d71968cbe5763b359fa7e77d09085f93ff97db2c24
10c4088b03d8ad96551a475587d934afbd2f468aecd9cc042016dae09fd7a66af6c4b936c35dd9c22a72ac7bf1fe27e81849941c10a59ceab
cac177c7b7b8e8b6d00cad6c991aed7bd01c22a51b141b9ca44c8239068a86a85920ec18c2f461593884d37825e91bb0662751675b3ce2973
69d5cd1f599ea931b3161de025b6b1ef1e8ad8ebbf5dc594e02929cc1dcdfc827b31e75342b663ee8b9d233bcdb8d5f867dec20becf2c48c4
ef9e43621f95e0c39a85e7e1f4c4cdc219ac71827f30fece1a0f042625cc907e53488bb725db6e17aa0149d80e37de1990573c5637b8ed496
```


Lastly, above is the API response file. As seen, the number of matches were 0.

In conclusion, our cookie extractor tool is different from other tools on the market because it extracts, analyzes cookies and then cross references them with a dark web database. In the future, we are hoping to enhance this tool by integrating machine learning to make the whole process more automated. Moreover, we are hoping to create a user-friendly graphical user interface. Additionally, we plan on making the tool a browser extension as well, for users hoping for an easier experience all around. Building on that idea we also plan on supporting mobile browsers as well.

References

Dark web API - 18 million posts and documents per month **(2024)** ***Twingly*****. Available at: https://www.twingly.com/dark-web-api/ (Accessed: 20 August 2024).**

Surveillance sweets: The hidden dangers of internet cookies **(2024)** ***NordVPN*****. Available at: https://nordvpn.com/research-lab/stolen-cookies-study/?srsltid=AfmBOor3oTzN0uQhHDRPyXSgqGbbSdFE0diSymoZbDBsOeUSVVNjxNMz (Accessed: 05 August 2024).**

Slama, R., Grubin, O. and Li, G. (2024) ***Catching compromised cookies*****,** ***Slack Engineering*****. Available at: https://slack.engineering/catching-compromised-cookies/ (Accessed: 31 August 2024).**