Colby Heilner

Professor Torres

5/3

IT 140

Lab 13

Part A:

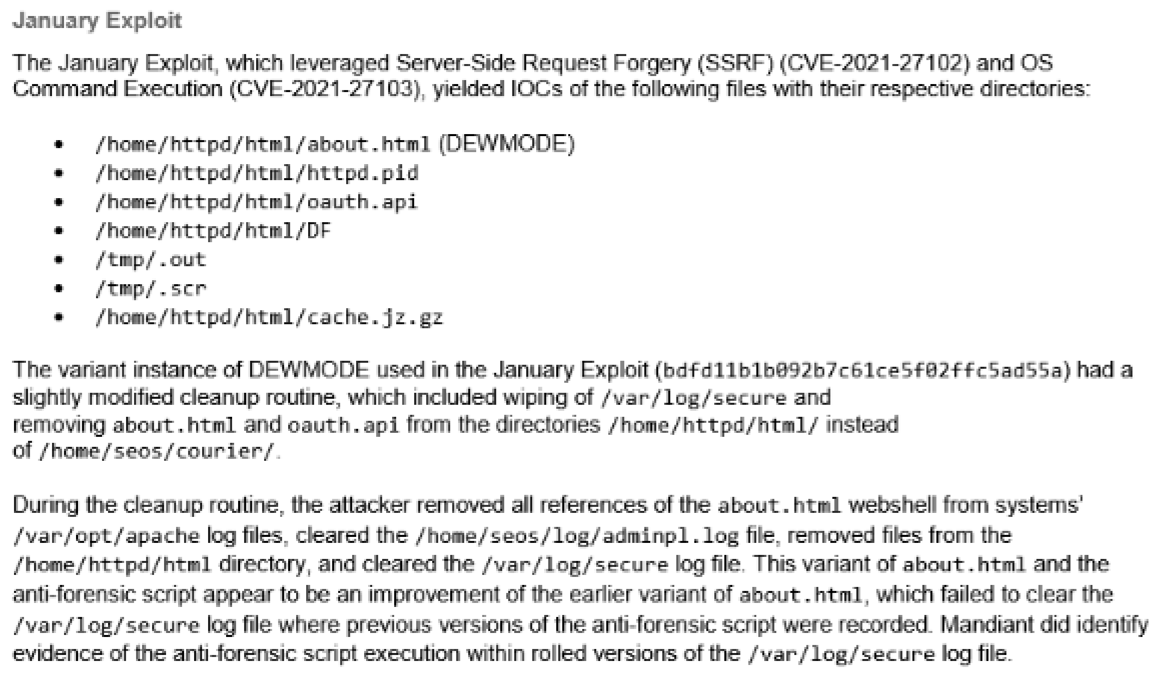
- Search the internet for penetration testing report. Find 5 sample Pentest reports (not vulnerability assessment reports) and answer the following questions on each report:
    - who is the report for
    - will upper management understand this report
    - will the technical staff fixing the issues understand the report
    - does the report provide sufficient information to remediate the problem
    - how would you adjust this report

Formatted with help from AI!

1. **FireEye (Mandiant) – Accellion FTA Security Assessment (2021)**

- **Client**: Accellion, Inc.
- **Audience**: Accellion's internal teams and stakeholders.
- **Purpose**: Security review of the FTA after exploitation.
- **Link**: Read report

  This one is slightly different as it goes through a review of events to determine how it was possible for attackers to have executed this attack. But it is easy to read, so I would say it is understandable by anyone who is slightly tech savvy.



January Exploit

The January Exploit, which leveraged Server-Side Request Forgery (SSRF) (CVE-2021-27102) and OS Command Execution (CVE-2021-27103), yielded IOCs of the following files with their respective directories:

- `/home/httpd/html/about.html` (DEWMODE)
- `/home/httpd/html/httpd.pid`
- `/home/httpd/html/oauth.api`
- `/home/httpd/html/DF`
- `/tmp/.out`
- `/tmp/.scr`
- `/home/httpd/html/cache.jz.gz`

The variant instance of DEWMODE used in the January Exploit (`bdfd11b1b092b7c61ce5f02ffc5ad55a`) had a slightly modified cleanup routine, which included wiping of `/var/log/secure` and removing `about.html` and `oauth.api` from the directories `/home/httpd/html/` instead of `/home/seos/courier/`.

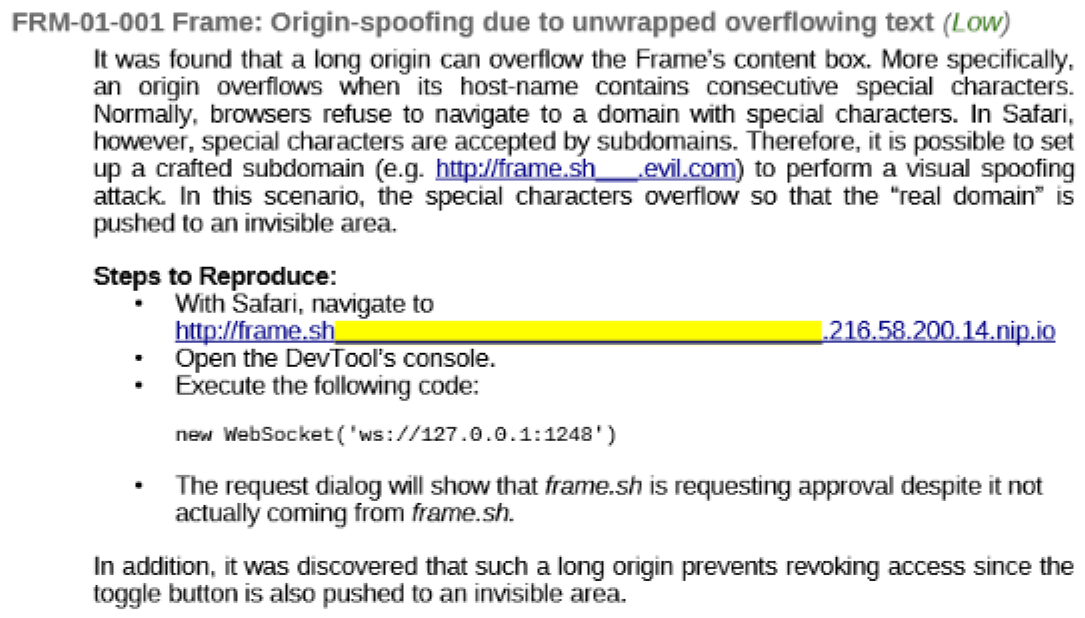
During the cleanup routine, the attacker removed all references of the `about.html` webshell from systems' `/var/opt/apache` log files, cleared the `/home/seos/log/adminpl.log` file, removed files from the `/home/httpd/html` directory, and cleared the `/var/log/secure` log file. This variant of `about.html` and the anti-forensic script appear to be an improvement of the earlier variant of `about.html`, which failed to clear the `/var/log/secure` log file where previous versions of the anti-forensic script were recorded. Mandiant did identify evidence of the anti-forensic script execution within rolled versions of the `/var/log/secure` log file.

I would add a timeline of compromise and patching activity.

2. **Cure53 – Frame Electron App Penetration Test (2018)**

- **Client**: Frame (Ethereum desktop wallet)
- **Audience**: Frame developers and security stakeholders
- **Purpose**: Assess the security of the Frame Electron-based desktop application
- **Link**: Read report

This one may be my favorite, and I will use this format for my reports. I really like how they use super clear steps to replicate. I think this would be super clear to anyone who knows what they are reading in the first place. It would also really help the admins and the devs to try and replicate the vulns to better learn to remediate them.



FRM-01-001 Frame: Origin-spoofing due to unwrapped overflowing text *(Low)*

It was found that a long origin can overflow the Frame's content box. More specifically, an origin overflows when its host-name contains consecutive special characters. Normally, browsers refuse to navigate to a domain with special characters. In Safari, however, special characters are accepted by subdomains. Therefore, it is possible to set up a crafted subdomain (e.g. http://frame.sh____.evil.com) to perform a visual spoofing attack. In this scenario, the special characters overflow so that the "real domain" is pushed to an invisible area.

Steps to Reproduce:

- With Safari, navigate to http://frame.sh______________________________.216.58.200.14.nip.io
- Open the DevTool's console.
- Execute the following code:

```
new WebSocket('ws://127.0.0.1:1248')
```

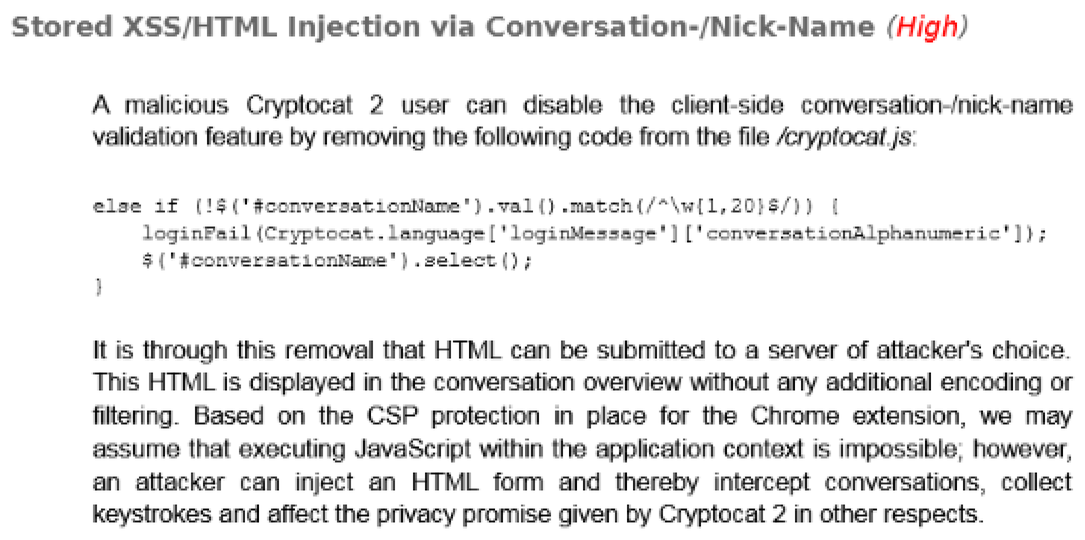
- The request dialog will show that *frame.sh* is requesting approval despite it not actually coming from *frame.sh*.

In addition, it was discovered that such a long origin prevents revoking access since the toggle button is also pushed to an invisible area.

3. **Cure53 – Cryptocat 2 Penetration Test**

- **Client**: Open Technology Fund
- **Audience**: Cryptocat developers & open-source community.
- **Purpose**: Security audit of encrypted chat app.
- **Link**: Read report

This one was a fun quick read, it was interesting to see a pen test on an anonymous messaging app. This vulnerability is crazy to have in this context!



Stored XSS/HTML Injection via Conversation-/Nick-Name *(High)*

A malicious Cryptocat 2 user can disable the client-side conversation-/nick-name validation feature by removing the following code from the file */cryptocat.js*:

```
else if (!$('#conversationName').val().match(/^\w{1,20}$/)) {
    loginFail(Cryptocat.language['loginMessage']['conversationAlphanumeric']);
    $('#conversationName').select();
}
```

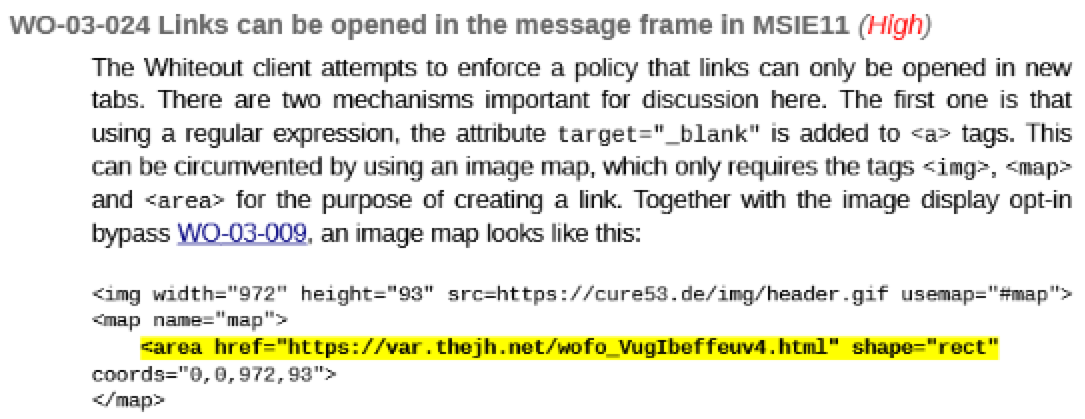
It is through this removal that HTML can be submitted to a server of attacker's choice. This HTML is displayed in the conversation overview without any additional encoding or filtering. Based on the CSP protection in place for the Chrome extension, we may assume that executing JavaScript within the application context is impossible; however, an attacker can inject an HTML form and thereby intercept conversations, collect keystrokes and affect the privacy promise given by Cryptocat 2 in other respects.

Overall, this is pretty easy to understand, and I would expect most admins and devs to be able to understand. I can see how higherups would of course get slightly lost. Lastly, I would Include more detailed remediation steps per vulnerability. Just to help out the staff a bit more because this is open-source software.

4. **Cure53 – Whiteout.io**

- **Client**: Whiteout.io
- **Audience**: Whiteout.io product and dev team.
- **Purpose**: Final pentest of their encrypted email app.
- **Link**: Read report

This one was slightly harder for me to read, the way that the lack of contrast in font size and color does not help. In terms of content, it is more technical, and I would not be surprised if a higherup would get bored reading this after a few seconds.



WO-03-024 Links can be opened in the message frame in MSIE11 *(High)*

The Whiteout client attempts to enforce a policy that links can only be opened in new tabs. There are two mechanisms important for discussion here. The first one is that using a regular expression, the attribute `target="_blank"` is added to `<a>` tags. This can be circumvented by using an image map, which only requires the tags `<img>`, `<map>` and `<area>` for the purpose of creating a link. Together with the image display opt-in bypass WO-03-009, an image map looks like this:

```
<img width="972" height="93" src=https://cure53.de/img/header.gif usemap="#map">
<map name="map">
    <area href="https://var.thejh.net/wofo_VugIbeffeuv4.html" shape="rect"
coords="0,0,972,93">
</map>
```

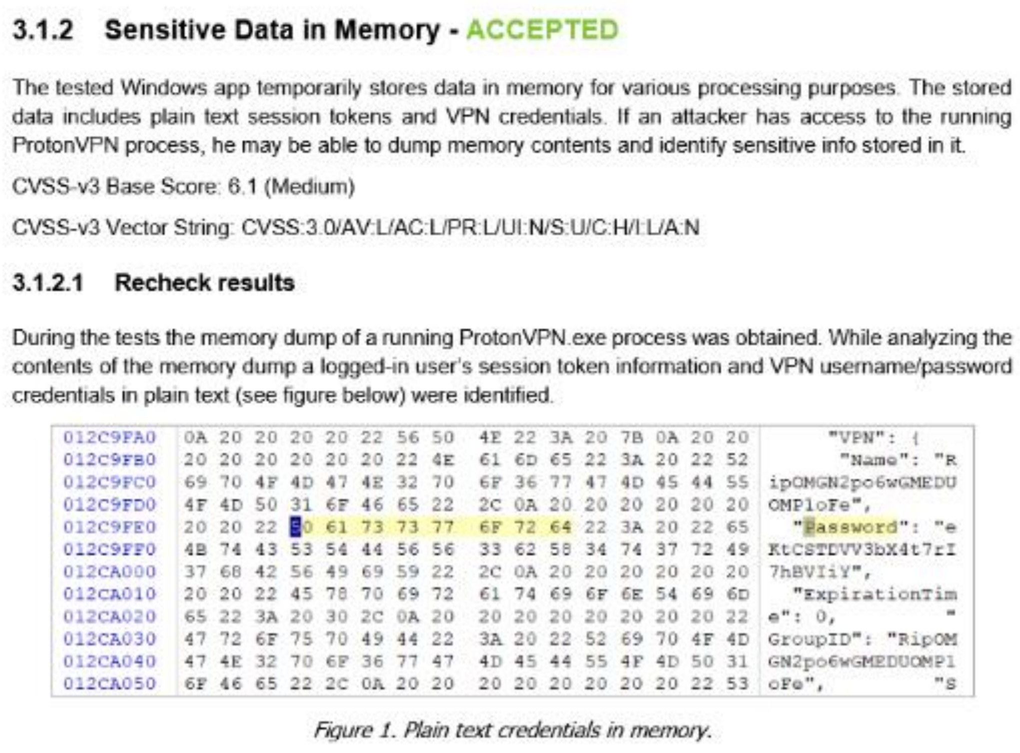
Now that being said a dev would understand this, and it's good for that case.

I would Break out issues by severity in a summary table. To give a clearer view at a glance.

5. **SEC Consult – ProtonVPN Windows App Audit (2020)**

- **Client**: Proton Technologies AG
- **Audience**: ProtonVPN development/security teams.
- **Purpose**: Pentest of ProtonVPN's Windows client.
- **Link**: Read report

    This is comprehendible by anyone! This included technical or not. Take this SS for example, they made effective use of more general words and explain how a password is shown in a dump of info on the machine running the software. They also who that the risk has been ACCEPTED, meaning they acknowledge it as most likely not worth fixing.



3.1.2 Sensitive Data in Memory - ACCEPTED

The tested Windows app temporarily stores data in memory for various processing purposes. The stored data includes plain text session tokens and VPN credentials. If an attacker has access to the running ProtonVPN process, he may be able to dump memory contents and identify sensitive info stored in it.

CVSS-v3 Base Score: 6.1 (Medium)

CVSS-v3 Vector String: CVSS:3.0/AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:L/A:N

3.1.2.1 Recheck results

During the tests the memory dump of a running ProtonVPN.exe process was obtained. While analyzing the contents of the memory dump a logged-in user's session token information and VPN username/password credentials in plain text (see figure below) were identified.

```
012C9FA0  0A 20 20 20 20 22 56 50  4E 22 3A 20 7B 0A 20 20       "VPN": {
012C9FB0  20 20 20 20 20 20 22 4E  61 6D 65 22 3A 20 22 52        "Name": "R
012C9FC0  69 70 4F 4D 47 4E 32 70  6F 36 77 47 4D 45 44 55  ipOMGN2po6wGMEDU
012C9FD0  4F 4D 50 31 6F 46 65 22  2C 0A 20 20 20 20 20 20  OMP1oFe",
012C9FE0  20 20 22 50 61 73 73 77  6F 72 64 22 3A 20 22 65    "Password": "e
012C9FF0  4B 74 43 53 54 44 56 56  33 62 58 34 74 37 72 49  KtCSTDVV3bX4t7rI
012CA000  37 68 42 56 49 69 59 22  2C 0A 20 20 20 20 20 20  7hBVIiY",
012CA010  20 20 22 45 78 70 69 72  61 74 69 6F 6E 54 69 6D    "ExpirationTim
012CA020  65 22 3A 20 30 2C 0A 20  20 20 20 20 20 20 20 22  e": 0,         "
012CA030  47 72 6F 75 70 49 44 22  3A 20 22 52 69 70 4F 4D  GroupID": "RipOM
012CA040  47 4E 32 70 6F 36 77 47  4D 45 44 55 4F 4D 50 31  GN2po6wGMEDUOMP1
012CA050  6F 46 65 22 2C 0A 20 20  20 20 20 20 20 20 22 53  oFe",          "S
```

Figure 1. Plain text credentials in memory.

Not sure what I would change like this one.

Part B:

- Run a full pentest against one target. Create a full report using our template. Conduct on ANY host, in ANY of the lab environments.

I will also assume it is okay to make a report on a TryHackMe machine, because I have done a few and it would be more interesting for me rather than netlab.