Gr@ve_Rose Follow
Dec 3, 2020 · 5 min read · Listen



# Two Systran Vulnerabilities and their Exploits

These are my first two independently discovered vulnerabilities which I've requested CVEs for (and will update the article if/when I get them) and am now at the point of agreed upon disclosure time with the vendor. Fair warning, these aren't mind-boggling CVSS 10.0 exploits which will render the Internet horribly insecure so if that's what you're expecting, I'm sorry to let you down. But if you're interested in what I've found as well as the mindset of hackers in general, please keep reading.

I was hired to perform a WebApp PenTest for a client against their installation of the Systran Pure Neural translation server. If you've never come across this application, you can enter text, upload documents or specify URLs to be translated from one language to another. Let's dig in…

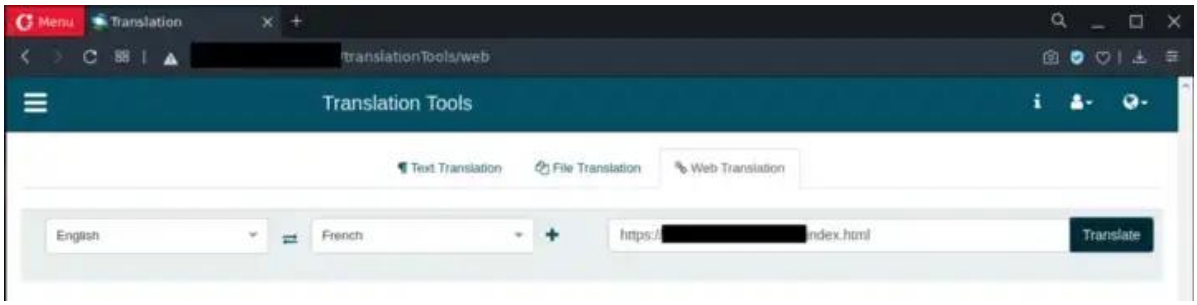
I was given an account on the Systran server as a regular user and, as always, began with reconnaissance by, well, just using the software. One of the most important tactics is understanding what the software is intended to do so that you're able to find where the limits are and attempt to get past those limits. I fired up Burp Suite and started to see what was happening as I translated some text. The first thing which jumped out at me was that there was an API key assigned to my account. The second thing is that the API key was in a "GET" request instead of a "POST" request which meant that the URI contained the API key. Looking at the address bar in my browser confirmed this. Of course I didn't need Burp to tell me this but I'm glad I had it running as I may not have noticed as quickly. I put my API key into my notes and kept looking around. Like most sites with any sort of authentication, I had a cookie assigned to me. Again, I copied that into my notes and kept on chooglin'.

Translating basic text didn't really lead to much so I started on the next tab: Uploading files. I once again started by putting a basic text file into the upload field with a phrase to be translated and watched Burp as it uploaded the file. The translation worked as expected so I created a new file. This time with some JavaScript in it:

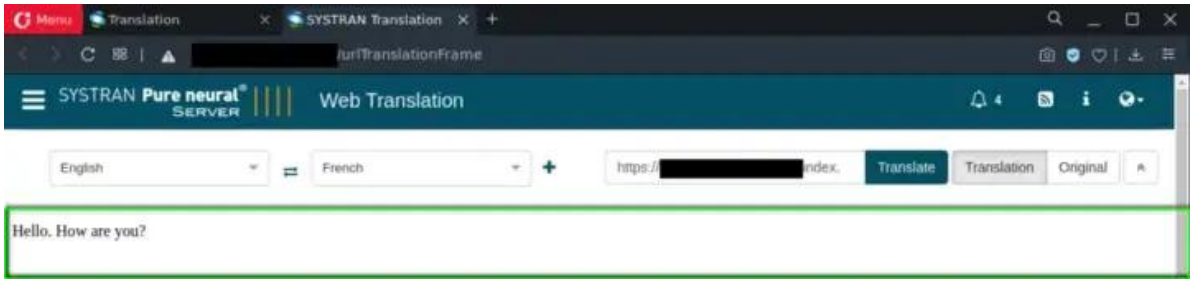
```
<script>alert(document.cookie);</script>
```

I uploaded the file and asked it to translate it for me but, sadly, my basic JavaScript didn't execute. I crafted a few more files both for server- and client-side attacks but they didn't yield any results. I was sure that this was going to be my first successful attack against the service or even, perhaps, a foothold into the back-end server itself but, alas, I was thwarted. Which, although rather sad for me as the attacker, is good for the client.

I plodded along to the next translation piece being the Web Translation feature of the application. Once more, I started with a simple translation by submitting a URL that I owned with a basic phrase and I clicked the Translate button. A new tab popped open with my translated phrase. Burp just showed that I was still communicating with the Systran server which made sense and there wasn't a whole lot to go after in the actual transmission of the data from the Systran server to my web browser.

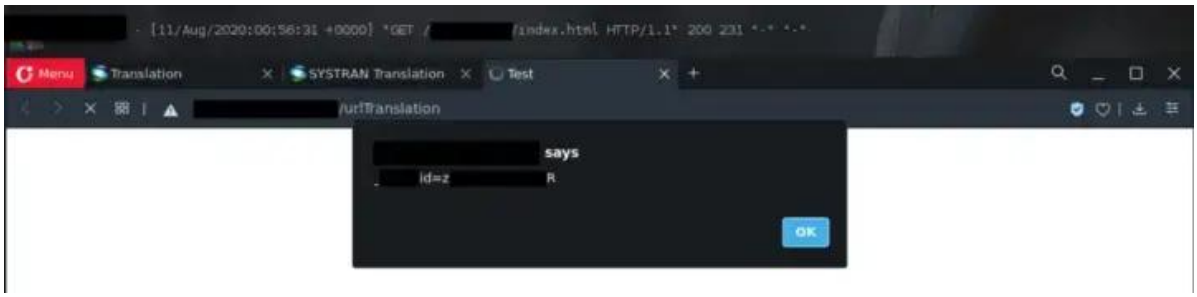



Systran Web Translation feature




Second tab opened

To my surprise, when I clicked the button to translate again from the *second* tab and the new tab popped open, the UI of the Systran software was absent. If my JavaScript didn't work inline, I wonder if it would work through XSS? I added the *<script>alert(document.cookie);</script>* into my hosted page and tried again from the start. Unfortunately, it didn't work when the second tab popped open. Well, since something was different on the third tab, why not try loading it again but this time, from the *second* tab?




Third tab executing XSS and log entry showing the Systran server loading (proxying) the connection

Boom. Exploit found. By having a user load a site for translation using the *second* tab popped open by an initial translation, a threat actor can interact with the user's browser directly. What's even better is that the client **never** connects to the attacking page directly; It's all proxied through the Systran server.

> *Update: CVE-2020–29539 has been assigned for the XSS exploit.*

Neat. I did promise two exploits so here's the other…

Remember that API key? As it turns out, there were no protections for limiting the target ports the Systran server can connect to nor were there any for rate-limiting. By using a valid API key and a bit of scripting, you can craft a basic HTTP/S scanner which the Systran server will proxy for you **and** you can use this as a reflective Denial-of-Service tool. Here are the key items (in BASH) for creating such tools:

- Create an array of targets (TGT) with protocol:target:port (http:example.net:80 https:www.google.com:443)
- Create a variable for your API key (API)
- Create a *for* loop to go through the array and create variables inside the loop for PROTO, TARGET and PORT which are set for each array item
- Run: *wget "hxxps://YOUR_SYSTRAN_SERVER/urlTranslation?source=en&target=fr&url=$PROTO%3A%2F%2F$TARGET%3A$PORT&owner=Systran&domain=Generic&size=M&apikey=$API&lang=en&noTranslation=1" — no-check-certificate -qO /dev/null*
- Check the return value ($?) and if it's "0" then that service port is open
- Want a reflected DoS instead? Same type of idea but use a "*while true*" style loop instead to send as many requests as possible since there's no rate limit or destination port checking (see Timeline below for the fixed version)

As you can see, by abusing the API calls, you can have the Systran server perform the Denial-of-Service attacks for you. In addition, you can use this as a targeted DoS against a service by specifying whatever port(s) you want inside your array; They don't have to be HTTP-type services — The Systran server will connect to any port you give it.

> *Update: CVE-2020–29540 has been assigned for the RDoS exploit.*

I want to thank Systran for accepting open discussion with me regarding both the vulnerabilities and exploits as well as understanding the responsible disclosure process. The people I interacted with were great to work with and had a penchant for understanding; Instead of being defensive, they took the time to ask questions, learn and address these issues in a very quick manner.

**Timeline**

August 20 — Vulnerabilities discovered and exploits created. Customer notified immediately.

September 2 — Call with Systran to notify them of the vulnerabilities.

September 9 — Systran releases version 9.7.0 which fixes the XSS and introduces a Rate Limiting feature to prevent the server from being used as a reflective Denial-of-Service proxy.

September 30 — Systran and I agreed that December 3rd would be the public disclosure date.

December 3 — Filed for CVE IDs for these two bugs. Published this document.

December 7 — CVE IDs provided in *Reservation* for these two security issues.

December 8 — CVE IDs 2020–29539 and 2020–29540 released from *Reservation* and are now public.