Security Platforms Engineering Coding Challenge

Overview

Thank you for your interest in Netflix and the role on the Security Platforms Engineering team! If you've made it this far, you should know that we are excited about you and are ready to further the mutual investment in seeing if there is a fit. The next step is a take-home coding challenge.

We have a take-home coding challenge because a work-sample is the best predictor of job performance and we want to see you at your best. You can do it at home with full access to the internet so you are in the coding environment you are comfortable with. The whole point of this is to have a fair and consistent evaluation of technical aptitude.

If you pass the coding challenge, the next step is the 'on-site' interview (virtual for now).

Security Platforms Engineering backend products are largely written in Python, and the day-to-day of engineering on the team is in Python. Because of that, we do prefer submissions to be made in Python. However, we expect that our stunning engineering colleagues are able to transfer programming skills to Python, so we will accept submissions in other languages. However, to ensure that we have the right background to correctly score the homework, please use a mainstream programming language (Python, Ruby, C, C++, C#, Go, Java, JavaScript.)

Your code will be reviewed by a member of our team who will score it against a rubric. We expect the challenge to take 4-6 hours to complete, and we expect to spend about 2 hours reviewing it.

If life gets in the way or you need to reschedule working on the challenge just drop me (jknecht@netflix.com) a line, and we'll reset expectations on timing.

If you have any questions or need more information about the challenge itself, please reach out to appseccodingpuzzle@netflix.com and we'll get back to you.

When you are done, please email a tgz/zip of your code to appseccodingpuzzle@netflix.com and we will get back to you within a few days.

Terminology

client account - unique numerical account used by the batch processor to run payment transactions. In the format of `/clients/<id>`.

Problem (hypothetical)

A long time ago, Netflix used a payments batch processing API for a number of client accounts. It had several useful features, one of them was a way to manually process a batch file by submitting a POST request with the date of the batch file to process. This was useful when the payment system failed and had to be jump started manually. This system was deprecated for a more reliable one but was kept online as a backup.

It's 9:37 AM and you are at the tail-end of a relatively tranquil on-call. You say "Wow it's been a quiet week!" Suddenly, your phone vigorously buzzes, vibrates, and voices that you are being paged! A researcher has been poking around legacy Netflix APIs and they have identified an "injection" vulnerability when testing out the batch processing API with a certain client account in the DEV environment. The report reads "*It looks like some client accounts have custom logic and a number of client accounts are unsafely parsing out the batch parameter to run a command line utility. I was able to exploit this vulnerability to retrieve the contents of a file '/etc/secret'.*

Your task is clear. You must identify all client accounts that are vulnerable to this injection vulnerability. You have access to the TEST instance of the batch API. There were once 500,000 client accounts used by the batch processor, however, the vast majority of those have been decommissioned. A team member gives you a log file at `https://appsec-exercise.test.netflix.net/log/client_access` of all the client ids that have been active in the last year. Phew! This seems to narrow down the search space significantly. She also tells you that there is only one batch file available for testing: 1-11-2015.txt.

Automate the exploit described below and scan every active client account to identify which client accounts are exploitable.

An unknown number of active clients are exploitable and contain different secrets. Find those machines and their secrets so we can patch this ASAP!

Payment Batch API Overview

The Batch API requires users to HMAC certain headers in order to facilitate batch processing. This is to ensure the user is authorized to run the batch processing function.

Affected Host

http://appsec-exercise.test.netflix.net

HMAC Key

`supersecretpassphrase`

Headers

The Payment API requires the following headers to be provided:

X-Netflix-AuthorizationTime

Example: 1429723538
This header has a current UTC timestamp rounded to the nearest second. Requests signed that are older than 5 minutes will expire. You should generate a new UTC timestamp with each request to avoid sending an expired header.

X-Netflix-PartnerName

Example: `foobar`
This header is static and lets the batch processor identify which secret key to use for signing. You must use foobar since we are only providing you this partner's key.

X-Netflix-Session

Example: `aec9ab75-90cd-4ef4-ab2c-4c6f9e79ce62`
It seems they are using this header to do some sort of anti-replay. You will need to ensure each request sets this field to a unique value (regardless of whatever host you are hitting). We suggest using a random identifier to ensure you won't hit duplicates but that choice is up to you. You should also set the same value in the **`cid`** parameter.

Warning
This header's value must match the body's **`cid`** parameter (see example request).

X-Netflix-HeaderSignature

Example:
`93c69e094a4eb7c13d8d849d1eca60f0ccc23b3888508bccb4cc6f9b5c46a2c7`
The HMAC signature, as described in the next section, for the previous three headers.

Note
Sending additional headers (aside from **`content-length`**) may cause problems with the API. No additional headers are needed for this exercise.

How to Sign Headers

1.) Convert each header into a `key=value` pair in alphabetical order and lower case
2.) Each **`key=value`** pair contains no whitespace and is comma delimited
3.) Sign using **`HMAC(<Credential>, <string-to-sign>)`**. Please use **`HMAC-SHA256`** for your algorithm.
4.) Signature needs to be hex encoded.
5.) Add the HTTP header **`X-Netflix-HeaderSignature`** with the hex signature as its value.

Example String to Sign

```
x-netflix-authorizationtime=1429723538,x-netflix-partnername=foobar,x-netflix-session=aec9ab75-90cd-4ef4-ab2c-4c6f9e79ce62
```

Signature Generation Pseudo Code

```
> sig = hmac("supersecretpassphrase", "x-netflix-authorizationtime=1429723538,x-netflix-partnername=foobar,x-netflix-session=aec9ab75-90cd-4ef4-ab2c-4c6f9e79ce62", sha256).hex

> print(sig)
93c69e094a4eb7c13d8d849d1eca60f0ccc23b3888508bccb4cc6f9b5c46a2c7
```

Exploit Payload

The injection exploit payload leverages the **`batch`** parameter and must exactly match the following:

```
batch=1-11-2015.txt; cat /etc/secret
```

Request Body Example

Combining the exploit payload and the necessary **`cid`** parameter yields a request body like the following:

`cid=aec9ab75-90cd-4ef4-ab2c-4c6f9e79ce62&batch=1-11-2015.txt; cat /etc/secret`

Example Request

You must use the **POST** method. You must POST the URLS listed in the file provided with the challenge.

example_request.txt

```
POST /clients/1 HTTP/1.1
Host: appsec-exercise.test.netflix.net
Content-Length: 77
X-Netflix-AuthorizationTime: 1429723538
X-Netflix-PartnerName: foobar
X-Netflix-Session: aec9ab75-90cd-4ef4-ab2c-4c6f9e79ce62
X-Netflix-HeaderSignature: 93c69e094a4eb7c13d8d849d1eca60f0ccc23b3888508bccb4cc6f9b5c46a2c7

cid=aec9ab75-90cd-4ef4-ab2c-4c6f9e79ce62&batch=1-11-2015.txt; cat /etc/secret
```

Important Notes

- Your code may take command line arguments, but you will be judged on how it runs with no arguments, so it should be optimized by default.
- Please do not maintain more than 10 open network connections with the service at the same time in your default solution.
- We've worked to make this exercise close to what our team does, and we'll be evaluating the code as though it was being committed to one of our projects - consider things like appropriate tests, and code readability.
- Both vulnerable and safe clients respond with 200 status codes.
- Be prepared to explain your solution to a member of the Application Security Engineering team.
- Use techniques to make your code reasonably fast since you have to scan thousands of endpoints (under 10 minutes receives full points, but some passing solutions can take longer).

- Don’t worry about breaking our application. If you observe consistent failures that don’t allow you to make progress, please reach out.
- Have fun!

Deliverables

Submit your work via email to appseccodingpuzzle@netflix.com. The email should include:

- A tgz/zip file containing:
  - Your working source code
  - A README file including instructions to build, if necessary, and run your scanner
- The output from a successful run of your scanner. Your no-argument scanner output must only include:
  - Start time
  - End time
  - Elapsed runtime in seconds
  - The number of clients that were active in the last year
  - The `id` of each client that was exploited and its associated secret
- Answers to the following:
  - How did you determine that you hit an exploitable client account? What clues did you observe?
  - How many vulnerable endpoints did you identify?
  - What technical or design decisions did you make in solving this challenge, and why?

Evaluation Criteria

Correctness and Robustness	10 points
Concurrency approach	5 points
Clarity of approach	4 points
General code quality	5 points
Testing	2 points
Subjective Catch-all	2 points