Software Security - Exercise sheet 2

Vianney HERVY

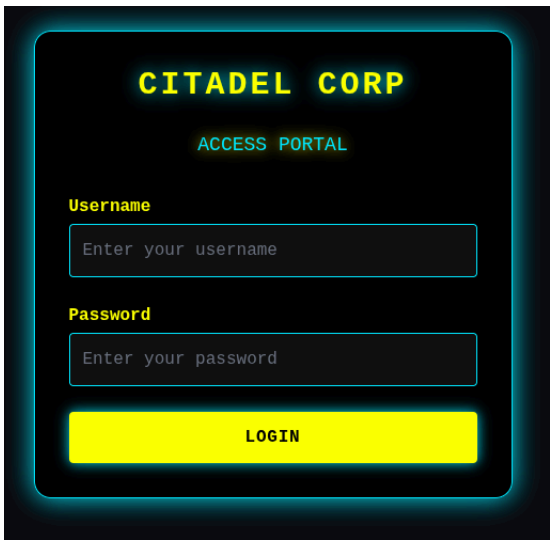
1. Web Application Vulnerabilites (Without Time Constraint)

I picked the niteCTF 2025[1] event. The below scoreboard can be found on the event's webpage.

320	TLS-7CTE	150.000	0.951
321	sully-vian	150.000	0.950
322	VoxSec	150.000	0.950

1.1. Database Reincursion

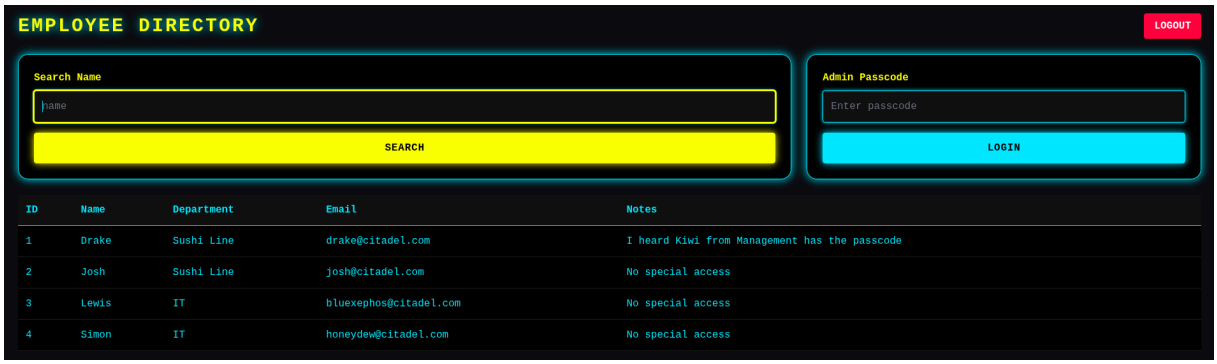
This challenge is in the "Web Exploitation" section. We are given a login page[2] with username and password field.



After some messing around I find that some inputs do not return "Invalid username or password", but "Input rejected by security filter". That happens specifically when my input contains `OR` or `--` both of which have meaning in SQL. I concluded there was a filter protecting the backend from SQL injection.

I looked up multiple other examples of SQL injection online until `' UNION SELECT 1,2,3/*` worked and brought me to the "Employee Directory" page.

[1] https://ctftime.org/event/2851
[2] https://database.chals.nitectf25.live/



There, I read “I heard Kiwi from Management has the passcode”. Looking up “Kiwi” shows 4 employees but none in management. Maybe the SQL query has a `LIMIT 4`. Indeed, inputting `'AND` breaks the qury and shows SQL error: unrecognized token: `' ORDER BY id LIMIT 4`. This message reveals the database is SQLite, and the exact structure of the query: `WHERE name = '<input>'ORDER BY id LIMIT 4`.

Having no access to the `OR` keyword, I tried using `IN` as `'IN (0,1) /*`, giving this: `WHERE name = ''IN (0,1) /*'ORDER BY id LIMIT 4`, effectively always being true (boolean is 0 or 1). But that only showed my the same users as the default page did. I needed to specify that I wanted the “Kiwi” user in the “Management” department: `Kiwi' AND department="Management"/*`. which makes the following request: `WHERE name = 'Kiwi' AND department="Management"/*'ORDER BY id LIMIT 4` that correctly shows the “Kiwi” user for Management with the admin passcode.

Entering the passcode shows the following new page:

Quick tries show that the input isn’t protected by the filter anymore. I can use `' OR 1 /*` to get all the reports quarters. I just need to extend the query to add elements from other tables. `' UNION SELECT * FROM metadata --` shows me interesting results:

ID	Quarter	Note	Revenue
1	reports	Quarterly revenue data for executives	quarter, note, revenue
2	users	Stores usernames and passwords	username, password
3	employees	Directory of employees and their notes	name, department, email, position, notes
4	metadata	Lists tables in this system	table_name, description, columns
5	CITADEL_ARCHIVE_2077	Restricted info (to be redacted by intern)	secrets

I now know the column names of the other tables and I discovered a new CITADEL_ARCHIVE_2077 which seems important. I crafted this input: `'UNION SELECT secrets,secrets,secrets,secrets FROM CITADEL_ARCHIVE_2077--` (secrets repeated 4 times to match the metadata table's column number). Unfortunately, this returns "Citadel SysSec: Query max length exceeded". I need to find a more concise query.

After looking up the SQL syntax again, I found out I can just write `'UNION SELECT secrets,1,1,1 FROM CITADEL_ARCHIVE_2077--` which is short enough and finally shows the flag.

1.2. Graph Grief[3]

This Challenge is in the "Web Exploitation" section. The provided hint is "Legacy XML importer may trigger internal file utilities"

Upon accessing the website, it was clear that the backend runs on Node.js, as indicated by the `x-powered-by: Express` header. The site advertised "AetherCorp" GraphQL API services. I attempted to access `/graphql` and found it to be an Apollo Server, but introspection was blocked.

Since introspection was disabled, I sent invalid queries to trigger error messages that might reveal the schema. For example, querying `{ usrs }` returned a suggestion: `Did you mean 'users' ?`. This allowed me to map out available fields: `users`, `profiles`, `orders`, `products`, and `node(id: ID!)`.

When querying `users`, I noticed a `role` field, with some users having the "admin" role. All IDs were base64 encoded, following the Relay-style global ID standard. Testing different IDs with the `node` query, I discovered a type called `secret`:

```
{ node(id: "c2VjcmV0OmZsYWc=") { ... on secret { id flag } } }
```

this returned

```
{ "data": { "node": { "id": "secret:flag", "flag": null } } }
```

[3]This challenge was solved in collaboration with another user, `vietrux`. Naturally, this section draws inspiration from his writeup: https://freebee.v1.io.vn/blog/nite-ctf-2025-writeup

The `secret:flag` node exists, but the flag value was null. The goal was to retrieve the actual flag value.

The hint suggested a legacy XML importer. I sent a request with `Content-Type: application/xml`:

```
<?xml version="1.0"?><root>hello</root>
```

The server echoed the text content, indicating XML parsing. I immediately considered XXE (XML External Entity) injection.

Basic XXE attempts were blocked:

- SYSTEM entities: "General SYSTEM entities are not allowed."
- Localhost access: "Localhost access via SYSTEM entity is not allowed."
- Only HTTP/HTTPS schemes were allowed.

I set up an external DTD server using Ngrok and tested if the server would fetch and parse external DTDs. It did, but the server also checked the content of the external DTD for blocked schemes like `file://`.

I bypassed the filter by using an absolute path (without `file://`) in the external DTD:

```
<!ENTITY % file SYSTEM "/etc/passwd">
<!ENTITY content "%file;">
```

This worked ! It allowed me to read `/etc/passwd`. I then located the flag by reading `flag.txt`, which contained the flag.

1.3. Floating-Point Guardian

This challenge is in the "AI" section. We are given a tcp connection[4] and the source of the code running on that server.

The program asks multiple questions that can be answered with a number (height, age, heart rate…) and writes these in an array. The array is then passed through a neural network. The output is compared to a secret target value. The goal is to approach that target value as close as possible.

```
$ ncat --ssl floating.chals.nitectf25.live 1337
I am the AI Gatekeeper.
Enter your details so I know you are my Master.
Answer these questions with EXACT precision...

[Q1]  What is your height in centimeters? 1
1
[Q2]  What is your weight in kilograms? 1
1
[Q3]  What is your age in years? 1
1
[Q4]  What is your heart rate (bpm)? 1
11
[Q5]  How many hours do you sleep per night?
```

[4] ncat –ssl floating.chals.nitectf25.live 1337

```
11
[Q6]  What is your body temperature in Celsius?
11
[Q7]  How many steps do you walk per day?
11
[Q8]  What is your systolic blood pressure?
11
[Q9]  How many calories do you consume daily?
11
[Q10] What is your BMI (Body Mass Index)?
11
[Q11] How many liters of water do you drink daily?
11
[Q12] What is your resting metabolic rate (kcal/day)?
11
[Q13] How many hours do you exercise per week?
11
[Q14] What is your blood glucose level (mg/dL)?
11
[Q15] Rate this CTF challenge out of 10:
11


Processing through neural network layers...
========================================
MASTER PROBABILITY: 0.9939367441
========================================

You are NOT the Master.
The neural network has rejected your identity.
```

The input is a 1x14 vector, so bruteforcing or groping towards the solution is out of the question.

Given that we have the source code, we can reproduce the neural network and optimize the input to minimize the offset. That's what I did, I ported the code from C to Python and used `differenctial_evolution` from `scipy.optimize` to bring the result's offset down to 10^{-11}.

I then wrote the results to the tcp connection which recognised me as the master and sent back the challenge's flag.

2. Known Real-World Software Vulnerabilities

2.1. Known vulnerabilities for the openssl product in 2025[5]

EUVD ID	CVE ID	CVSS	Summary
EUVD-2025-31727	CVE-2025-9232	5.9	An application using the OpenSSL HTTP client API functions may trigger an out-of-bounds read if the 'no_proxy' environment variable is …
EUVD-2025-31728	CVE-2025-9231	6.5	A timing side-channel which could potentially allow remote recovery of the private key exists in the SM2 algorithm implementation on 64…

[5]Source: EUVD's public API

EUVD ID	CVE ID	CVSS	Summary
EUVD-2025-31729	CVE-2025-9230	7.5	An application trying to decrypt CMS messages encrypted using password based encryption can trigger an out-of-bounds read and write. I…
EUVD-2024-51400	CVE-2024-13176	4.1	A timing side-channel which could potentially allow recovering the private key exists in the ECDSA signature computation. Impact summa…
EUVD-2025-16128	CVE-2025-4575	6.5	Use of -addreject option with the openssl x509 application adds a trusted use instead of a rejected use for a certificate. Impact summ…
EUVD-2024-51115	CVE-2024-12797	6.3	Clients using RFC7250 Raw Public Keys (RPKs) to authenticate a server may fail to notice that the server was not authenticated, because…

I selected the **EUVD-2025-16128** vulnerability[6] for the rest of the exercise:

> Issue summary: Use of -addreject option with the openssl x509 application adds a trusted use instead of a rejected use for a certificate.
>
> Impact summary: If a user intends to make a trusted certificate rejected for a particular use it will be instead marked as trusted for that use.
>
> A copy & paste error during minor refactoring of the code introduced this issue in the OpenSSL 3.5 version. If, for example, a trusted CA certificate should be trusted only for the purpose of authenticating TLS servers but not for CMS signature verification and the CMS signature verification is intended to be marked as rejected with the -addreject option, the resulting CA certificate will be trusted for CMS signature verification purpose instead.
>
> Only users which use the trusted certificate format who use the openssl x509 command line application to add rejected uses are affected by this issue. The issues affecting only the command line application are considered to be Low severity.
>
> The FIPS modules in 3.5, 3.4, 3.3, 3.2, 3.1 and 3.0 are not affected by this issue.
>
> OpenSSL 3.4, 3.3, 3.2, 3.1, 3.0, 1.1.1 and 1.0.2 are also not affected by this issue.

2.2. Source code vulnerability

The vulnerability comes from the `openssl x509` cli application. Specifically, the logic handling the `-addreject` flag was reversed.

[6] https://euvd.enisa.europa.eu/vulnerability/EUVD-2025-16128

[7] https://github.com/openssl/openssl/blob/1b20579d5c35ca8d3c6e79fb6f5067ad98c47beb/apps/x509.c#L460

The following snippet comes from the `x509.c` file at the `1b20579` commit[7]. It is part of a switch statement which dispatches the handling depending on the current cli argument. Line `468` explicitely attempts to push `objtmp` to the `trust` stack `reject` stack.

```
460         case OPT_ADDREJECT:
461             if (reject == NULL && (reject = sk_ASN1_OBJECT_new_null()) == NULL)
462                 goto end;
463             if ((objtmp = OBJ_txt2obj(opt_arg(), 0)) == NULL) {
464                 BIO_printf(bio_err, "%s: Invalid reject object value %s\n",
465                            prog, opt_arg());
466                 goto opthelp;
467             }
468             if (!sk_ASN1_OBJECT_push(trust, objtmp))
469                 goto end;
470             trustout = 1;
471             break;
```

2.3. Vulnerability fix

The fix is simple and can be shown in this GitHub commit[8] which also adds tests.

It involves changing the stack poinyer from `trust` to `reject` to ensure the certification use is correctly categorized as rejected.

```
                           prog, opt_arg());
                goto opthelp;
            }
-           if (!sk_ASN1_OBJECT_push(trust, objtmp))
+           if (!sk_ASN1_OBJECT_push(reject, objtmp))
                goto end;
            trustout = 1;
            break;
```

2.4. Type of vulnerability

This vulnearbility is a **CWE-480: Use of Incorrect Operator**[9]. It is described as *"The product accidentally uses the wrong operator, which changes the logic in security-relevant ways."* and *"These types of errors are generally the result of a typo by the programmer."* which matches this exact case of copy-paste error.

2.5. What can be learned

Even minor refactoring can induce critical security flaws if code blocks are duplicayed and not carefully adapted to their new context.

Each command-line flag should have a specific test case that verifies it is correctly handled and applied. In this case, a test should have checked if the resulting certificate actually contained a "Rejected Use" extension.

This kind of bugs can't be caught by static analysis tools given that the code "works" but does the opposite of what was intended. On the other hand, they are easily caucht by functional testing and especially regression testing that would check the refactor didn't break anything.

[8] https://github.com/openssl/openssl/commit/e96d22446e633d117e6c9904cb15b4693e956eaa
[9] https://cwe.mitre.org/data/definitions/480.html

This vulnerability is very specific to the cli tool users, of the version 3.5.0 and with the `-addreject` flag. In fact, it has a low CVSS score (6.5) and a low EPSS score (0.01%).