No rate limiting & Logic flaws

A04:2021-Insecure Design

Insecure Design meaning

The app is badly designed from the start, even if code is “clean”.

Routes and files

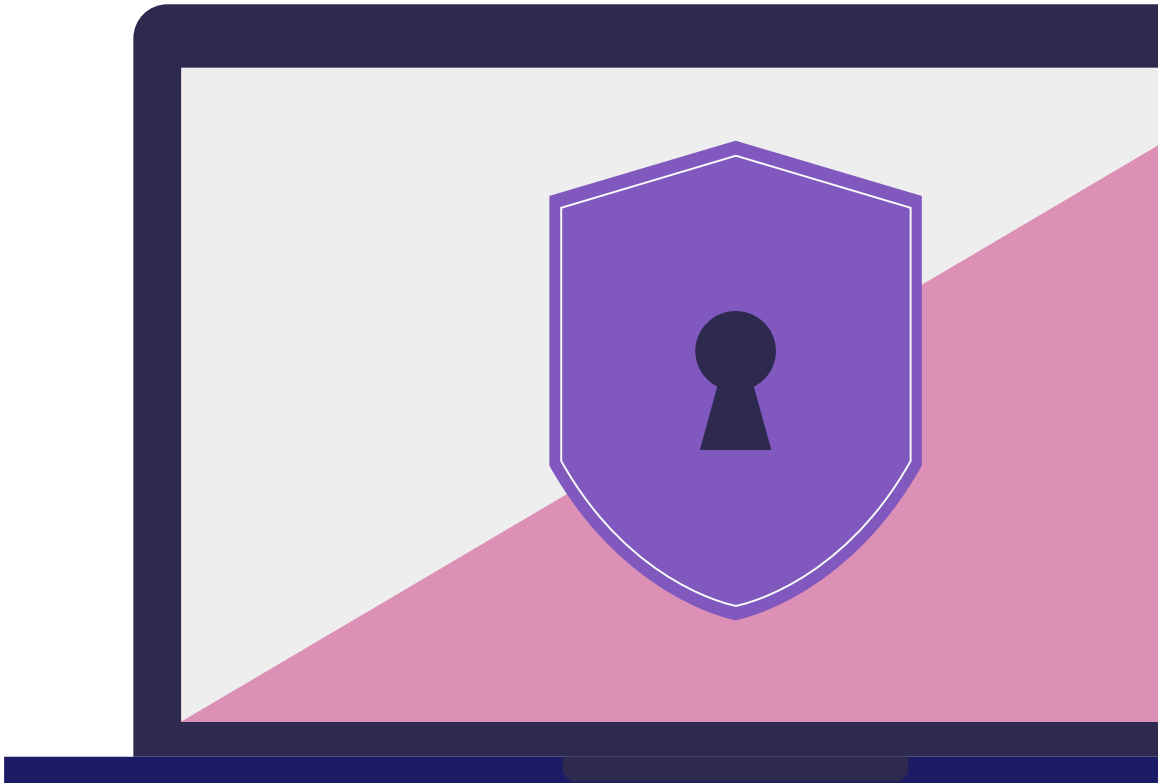
Files

- frontend.js
- order.js

Routes

- /login
- /registerform
- /v1/search/:filter/:query

No rate limiting



vulnerable code

```
app.get('/v1/search/:filter/:query', (req,res) =>{
    const filter = req.params.filter
    const query = req.params.query
        const sql = "SELECT * FROM beers WHERE "+filter+" = '"+query+"'";

        const beers = db.sequelize.query(sql, { type: 'RAW' }).then(beers => {
            res.status(200).send(beers);
        }).catch(function (err) {
            res.status(501).send("error, query failed: "+err)
          })

});
```

exploitation

```
for i in {1..1000}; do curl -s "http://localhost:5000/v1/search/id/$i" & done
```

---

explanation

The /v1/search endpoint has no rate limiting. This missing architecture allows attackers to overwhelm the server and database by sending thousands of rapid requests.
An attacker can execute a DoS attack by spawning many concurrent queries, exhausting database connections, or scrape all data by systematically querying every possible ID without restriction.

fixes

```
app.get('/v1/search/:filter/:query', (req,res) =>{
    const filter = req.params.filter
    const query = req.params.query
        const sql = "SELECT * FROM beers WHERE "+filter+" = '"+query+"'";

        const beers = db.sequelize.query(sql, { type: 'RAW' }).then(beers => {
            res.status(200).send(beers);
        }).catch(function (err) {
            res.status(501).send("error, query failed: "+err)
          })

});
```

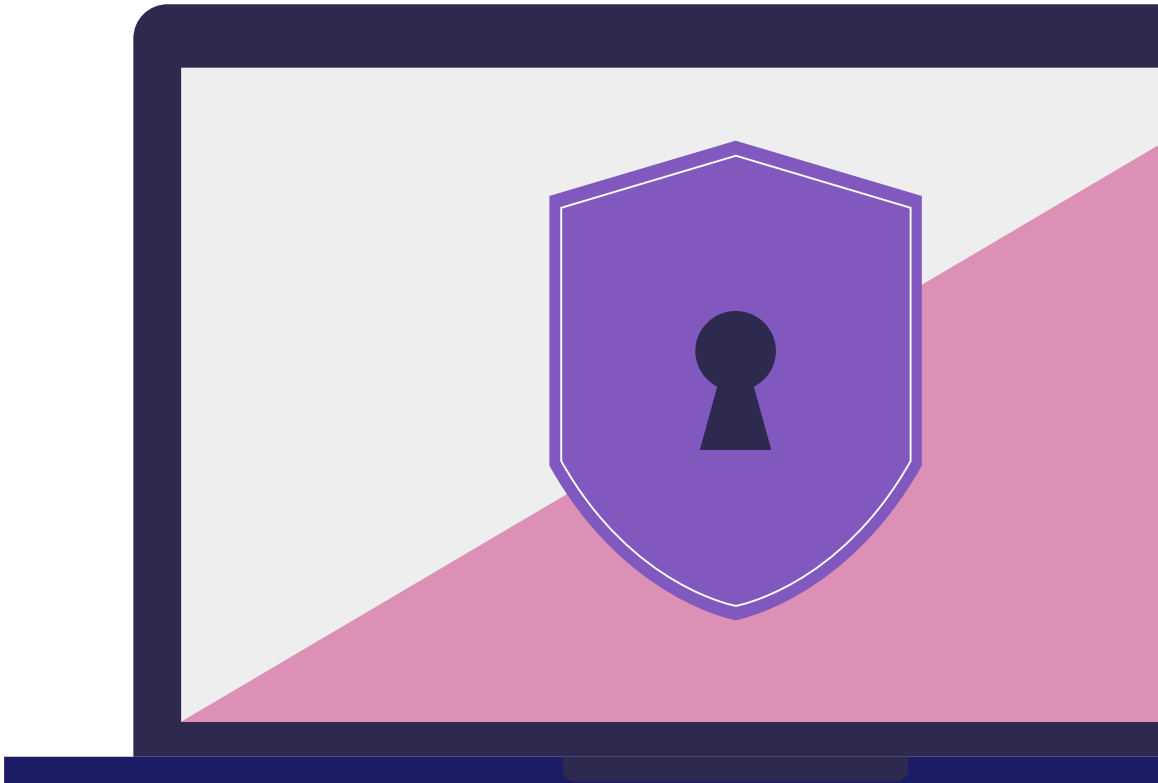
```
const rateLimit = require('express-rate-limit');
const searchLimiter = rateLimit({
  windowMs: 1 * 60 * 1000,
  max: 10,
  message: { error: 'Too many search requests, please try again later.' },
  standardHeaders: true,
  legacyHeaders: false,
});

app.get('/v1/search/:filter/:query', searchLimiter, (req,res) =>{
    const filter = req.params.filter
    const query = req.params.query
    const sql = "SELECT * FROM beers WHERE "+filter+" = '"+query+"'";
    const beers = db.sequelize.query(sql, { type: 'RAW' }).then(beers => {
        res.status(200).send(beers);
    }).catch(function (err) {
        res.status(501).send("error, query failed: "+err)
    })
});
```

Semgrep rule

Static rules cannot detect a missing security control like rate limiting. A04 vulnerabilities are gaps in the design,not bugs in the code, making them invisible to tools that scan for specific bad patterns.

Logic flaws



vulnerable code

```
if((user[0].password == userPassword) || (md5(user[0].password) == userPassword)){
    req.session.logged = true
    res.redirect('/profile?id='+user[0].id);
    return;
}
```

explanation

The authentication endpoint contains a critical logic flaw that completely breaks legitimate user access. The condition:

if((user[0].password == userPassword) || (md5(user[0].password) == userPassword))

is designed to accept either a plaintext password or its MD5 hash, but because passwords are stored as MD5 hashes, the logic becomes impossible to satisfy. The first condition compares a stored hash (like 21232f297a57a5a743894a0e4a801fc3) with a plaintext user input (like admin), which is always false. The second condition makes the error worse by hashing the already-stored hash and comparing that to the plaintext input, creating a comparison like md5("21232f297a57a5a743894a0e4a801fc3") == "admin", which is also always false. This flawed design not only prevents all valid logins through this endpoint, rendering it useless for users, but also represents a fundamental A04:2021-Insecure Design vulnerability where the authentication logic itself is architecturally broken.

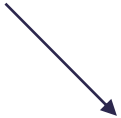
---

exploitation

you can use the hash of the password to login

fixes

```
if((user[0].password == userPassword) || (md5(user[0].password) == userPassword)){
    req.session.logged = true
    res.redirect('/profile?id='+user[0].id);
    return;
}
```



```
if (user[0].password === md5(userPassword)) {
    req.session.logged = true
    res.redirect('/profile?id='+user[0].id);
    return;
}
```

Semgrep rule

Since this is a logic error in how the code thinks not a specific mistake in how it's written, you can't write a simple semgrep rule to find it.
These mistakes can be anywhere and look like normal code so they slip right past automated security checks.
This means you have to carefully review the design and think through the logic yourself