

Wybrane elementy praktyki projektowania oprogramowania  
Wykład 09/15  
node.js: Express (2)

Wiktor Zychła 2018/2019

---

1	Spis treści	
2	Silne typowanie argumentów funkcji w VS Code .....	2
3	Domyślne middleware do obsługi nieobsłużonych ścieżek oraz zagrożenie Cross-site scripting... 3	
4	Użycie parametrów w ścieżkach i zagrożenie Web Parametr Tampering.....	4
5	Obsługa ciastek.....	6
6	Obsługa kontenera sesji na serwerze .....	9
7	Złożone szablony z parametrami .....	11

## 2 Silne typowanie argumentów funkcji w VS Code

Na przykładzie funkcji typu middleware można zademonstrować sposób na uzyskanie efektu podpowiadania typu obiektu. W tym celu należy użyć elementów TypeScript (**import**) oraz napisać komentarz w składni [JSDoc](#).

```
2 import * as http from "http" // zaremovać przed uruchomieniem
3
4 /**
5  *
6  * @param {http.IncomingMessage} req
7  * @param {http.ServerResponse} res
8  * @param {*} next
9  */
10 function middleware(req, res, next) {
11   req. // <- tu działa podpowiadanie składni
12 }
```

Tego sposobu można użyć do dowolnego typu argumentów

```
/**
 * @param {number} n
 * @param {string} s
 */
function foo(n, s) {
  s.
}
```

### 3 Domyślne middleware do obsługi nieobsłużonych ścieżek oraz zagrożenie Cross-site scripting

„Domyślne” middleware, dodane jako ostatnie, będzie obsługiwać wszystkie nieobsłużone do tej pory ścieżki. Można użyć tej techniki do przechwycenia żądań do nieobsługiwanych ścieżek:

```
// ... wcześniej inne mapowania ścieżek

app.use((req, res, next) => {
  res.render('404.ejs', { url : req.url });
});

http.createServer(app).listen(3000);
```

```
<!-- 404.ejs -->
<html>
<body>
  Strona <%= url %> nie została znaleziona.
</body>
</html>
```

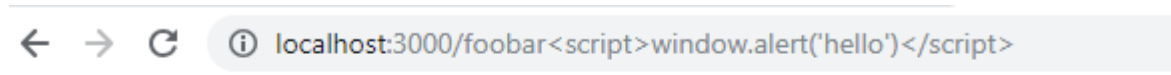
Przy okazji przyjrzymy się temu że wartość **req.url** jest w obiekcie żądania zakodowana w standardzie [Percent-encoding](#) (URL Encoding), co ma chronić aplikację przed prostym atakiem [Script injection](#) – a konkretnie jego wersją nazwaną [Cross-site scripting](#).

W tym ataku, atakujący może spreparować jakiś zasób na serwerze, który po odwiedzeniu go przez atakowanego spowoduje wykonanie dowolnego skryptu Javascript po stronie przeglądarki. Daje to atakującemu możliwość wykradania wartości wpisanych do formularza i odsyłania ich na kontrolowane przez niego serwery.

Tu: możliwy atak polegałby na przesłaniu przez atakującego spreparowanego odnośnika zawierającego fragment skryptu do wykonania:

[http://localhost:3000/foobar<script>window.alert\('hello'\)</script>](http://localhost:3000/foobar<script>window.alert('hello')</script>)

W zaprezentowanej powyżej wersji taki odnośnik spowoduje wyrenderowanie nieczytelnego



Strona /foobar%3Cscript%3Ewindow.alert('hello')%3C/script%3E nie została znaleziona.

Wystarczyłoby jednak w kodzie użyć funkcji **decodeURIComponent** i równocześnie w widoku użyć **<%= %>** zamiast **<%= url %>**, aby otworzyć podatność. Jej źródłem jest bezrefleksyjne zwrócenie użytkownikowi zawartości, której część pochodzi od niego samego (lub innego użytkownika).

## 4 Użycie parametrów w ścieżkach i zagrożenie Web Parametr Tampering

Możliwe jest dynamiczne parametryzowanie ścieżek

```
var http = require('http');
var express = require('express');

var app = express();

app.set('view engine', 'ejs');
app.set('views', './views');

app.use(express.urlencoded({extended:true}));

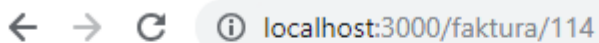
app.get("/",
  (req, res) => { res.end("default page");});

app.get("/faktura/:id",
  (req, res) => { res.end(`dynamicznie generowana faktura:
${req.params.id}`)});

// ... wcześniej inne mapowania ścieżek

app.use((req, res, next) => {
  res.render('404.ejs', { url : req.url });
});

http.createServer(app).listen(3000);
```

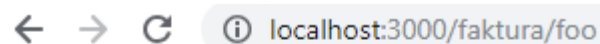


← → ↻ ⓘ localhost:3000/faktura/114

dynamicznie generowana faktura: 114

Taka ścieżka dopasowuje się do całej klasy ścieżek, a ograniczenie dopasowania polega na możliwości użycia wyrażenia regularnego, na przykład wymuszającego tylko liczby

```
app.get("/faktura/:id(\\d+)",
  (req, res) => { res.end(`dynamicznie generowana faktura:
${req.params.id}`)});
```



← → ↻ ⓘ localhost:3000/faktura/foo

Strona /faktura/foo nie została znaleziona.

Użycie parametrów w ścieżkach otwiera aplikację na kolejny typ podatności o którym warto wspomnieć, tzw. [Web Parameter Tampering](#) (Query string tampering). Atak ten polega na tym że pasek adresowy jest pod kontrolą użytkownika przeglądarki, a programiście aplikacji webowej zdarza się o tym zapomnieć.

Typowa sytuacja w której dochodzi do podatności polega na sytuacji w której aplikacja dla konkretnego użytkownika generuje link do konkretnego zasobu (faktury, powiadomienia, itp.) i wysyła taki link użytkownikowi innym kanałem (np. e-mail czy sms). Użytkownik po otwarciu linka dostaje się do zasobu przewidzianego dla niego, ale modyfikując pasek adresowy może mieć dostęp do innych zasobów.

Jednym z typowych mechanizmów ochrony przez tym zagrożeniem jest użycie dodatkowego parametru weryfikującego poprawność odnośnika, wykorzystującego mechanizm HMAC ([Hash Message Authentication Code](#)). Na serwerze, parametr ścieżki jest łączony z kluczem tajnym a otrzymana wartość jest przepuszczana przez jednokierunkową funkcję skrótu (np. SHA2). Wynik jest dodawany do adresu jako jego „podpis”:

<http://localhost/faktura/114?mac=7658765876587658765abfe789789>

Przy przetwarzaniu żądania na serwerze operacja wyliczenia „podpisu” jest powtarzana, a niezgodność podpisu wyliczonego z oczekiwanym oznacza że użytkownik **zmodyfikował** wartość parametru. Taką sytuację można jawnie obsłużyć i zwrócić komunikat błędu.

## 5 Obsługa ciastek

Do obsługi ciasteczek służy middleware **cookie-parser**. Wartość ciastka utworzona na serwerze jest dodawana do odpowiedzi w nagłówku **Set-cookie**, a następnie dołączana przez przeglądarkę w każdym żądaniu. Na serwerze można odczytać wartości przychodzących ciastek:

```
var http = require('http');
var express = require('express');
var cookieParser = require('cookie-parser');

var app = express();

app.set('view engine', 'ejs');
app.set('views', './views');

app.disable('etag');
app.use(cookieParser());
app.use(express.urlencoded({
  extended: true
}));

app.use("/", (req, res) => {
  var cookieValue;
  if (!req.cookies.cookie) {
    cookieValue = new Date().toString();
    res.cookie('cookie', cookieValue);
  } else {
    cookieValue = req.cookies.cookie;
  }

  res.render("index", { cookieValue: cookieValue });
});

http.createServer(app).listen(3000);
```

```
<!-- views/index.ejs -->
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>

<body>
  <form method="POST">
    Wartość z ciastka: <%= cookieValue %>
```

```

    <button>Zapisz</button>
  </form>
</body>

</html>

```

Spośród możliwych parametrów ciastka interesują nas

- **maxAge** umożliwiające sterowanie czasem życia ciastka w przeglądarce, w tym usunięcie ciastka (maxAge: -1)
- **signed** dodające do ciastka podpis HMAC wygenerowany z klucza dostarczonego jako argument funkcji **cookieParser()** - dostęp do podpisanych ciastek wymaga odwołania się do właściwości **signedCookies** obiektu **request**

```

var app = express();

app.set('view engine', 'ejs');
app.set('views', './views');

app.disable('etag');
app.use(cookieParser());
app.use(express.urlencoded({
  extended: true
}));

app.use("/", (req, res) => {
  var cookieValue;
  if (!req.cookies.cookie) {
    cookieValue = new Date().toString();
    res.cookie('cookie', cookieValue, {
  } else {
    cookieValue = req.cookies.cookie;
  }

  res.render("index", { cookieValue: cookieValue });
});

http.createServer(app).listen(3000);

```

cookie(name: string, val: string, options: CookieOptions): Response

Set cookie `name` to `val`, with the given `options`.

Options:

- `maxAge`` max-age in milliseconds, converted to `expires``
- `signed`` sign the cookie
- `path`` defaults to `"/"`

Examples:

```
// "Remember Me" for 15 minutes
```

- domain?
- encode?
- expires?
- httpOnly?
- maxAge?
- path?
- sameSite?
- secure?
- signed?
- app
- cookie
- cookieParser

```

var http = require('http');
var express = require('express');
var cookieParser = require('cookie-parser');

var app = express();

app.set('view engine', 'ejs');
app.set('views', './views');

app.disable('etag');
app.use(cookieParser('xzufybuixyfbuxziyfbuzixfuyb'));
app.use(express.urlencoded({
  extended: true
}));

```

```
});  
  
app.use("/", (req, res) => {  
  var cookieValue;  
  if (!req.signedCookies.cookie) {  
    cookieValue = new Date().toString();  
    res.cookie('cookie', cookieValue, { signed: true });  
  } else {  
    cookieValue = req.signedCookies.cookie;  
  }  
  
  res.render("index", { cookieValue: cookieValue } );  
});  
  
http.createServer(app).listen(3000);
```



## 6 Obsługa kontenera sesji na serwerze

Kontener sesji to zbiornik na dane po stronie serwera, który znosi ograniczenie rozmiaru ciastek – zamiast transferować cały stan do użytkownika, wysyła mu się klucz w kontenerze, a stan zapamiętuje na serwerze pod kluczem. Za obsługę sesji odpowiada middleware [express-session](#).

Co ważne – architektura sesji zakłada możliwość użycia na serwerze *dostawcy* kontenera, którym może być np. [zewnętrzna baza danych](#).

```
var http = require('http');
var express = require('express');
var session = require('express-session');

var app = express();

app.set('view engine', 'ejs');
app.set('views', './views');

app.disable('etag');

app.use(session({resave:true, saveUninitialized: true, secret:
'qewhiugriasy'}));

app.use("/", (req, res) => {
  var sessionValue;
  if (!req.session.sessionValue) {
    sessionValue = new Date().toString();
    req.session.sessionValue = sessionValue;
  } else {
    sessionValue = req.session.sessionValue;
  }

  res.render("index", { sessionValue: sessionValue });
});

http.createServer(app).listen(3000);
```

```
<!-- views/index.ejs -->
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>

<body>
```

```
<form method="POST">
  Wartość z session: <%= sessionValue %>
  <button>Zapisz</button>
</form>
</body>
</html>
```

## 7 Złożone szablony z parametrami

Istnieje możliwość wywołania szablonu z innego szablonu

```
<!-- views/index.ejs -->
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>

<body>
  Wywołanie szablonu z innego szablonu:
  <% var name='combo1' %>
  <% var options= [
    { value : 1, text : 'element 1' },
    { value : 2, text : 'element 2' },
    { value : 3, text : 'element 3' }
  ]
  %>
  <% include select %>
</body>

</html>
```

```
<!-- select.ejs -->
<select name='<%= name %>'>
  <% options.forEach( option => { %>
  <option value='<%= option.value %>'>
    <%= option.text %>
  </option>
  <% }) %>
</select>
```

← → ↻ ⓘ localhost:3000

Wywołanie szablonu z innego szablonu: