Makalah Tugas Kuliah EL5215

Keamanan Perangkat Lunak

KEAMANAN PERANGKAT LUNAK PADA BAHASA PEMROGRAMAN NODE.JS UNTUK APLIKASI BERBASIS *WEB*

Oleh :

MUHAMMAD LUQMAN

23215127

Dosen :

Dr. Ir. Budi Rahardjo

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

PROGRAM MAGISTER TEKNIK ELEKTRO

INSTITUT TEKNOLOGI BANDUNG

2016

ABSTRAK

Pengembangan aplikasi berbasis *web* saat ini sedang mengalami peningkatan. Berbagai macam bahasa pemrograman dapat digunakan untuk membuat suatu *web* aplikasi, salah satunya adalah Node.js. Node.js dieksekusi sebagai aplikasi *server*. Platform ini menggunakan bahasa pemrograman *javascript* dan menggunakan teknik *non-bloking* untuk mempercepat proses. Komunitas Node.js pada umumnya sedang fokus terhadap skalabilitas platform Node.js, dan penelitian terkait keamanan aplikasi berbasis Node.js juga sedang berkembang pesat. Makalah ini menguraikan beberapa aspek keamanan yang harus diperhatikan ketika menggunakan platform yang Node.js dan sisi *server* JavaScript . Dijelaskan pula teknik untuk mengembangkan dan mengkonfigurasi aplikasi web yang aman dan tangguh pada *platform* Node.js .

Kata kunci : Node.js, *web* aplikasi, keamanan aplikasi

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

Bab I Pendahuluan

Perkembangan teknologi dalam hal penyebaran informasi telah sangat maju di era tahun 2000-an. Salah satu sarana untuk penyebaran informasi adalah media elektronik yang disebarkan melalui internet. Perkembangan internet yang berkembang pesat mendorong pengembangan layanan *web* yang *real-time*. Layanan *web* yang *real-time* adalah layanan yang membuat pengguna menerima informasi pada waktu yang sama saat penulis menerbitkan informasi pada layanan *web* tersebut.

Salah satu teknik pengembangan *web* aplikasi adalah dengan adanya hubungan antara *client* dan *server,* yang biasa disebut *front-end* dan *back-end*. Untuk mengembangkan sebuah *web server*, telah banyak digunakan bahasa pemrograman untuk melakukannya. Salah satu pengembangan dalam *web server* adalah Node.js. Node.js dikembangkan mulai awal tahun 2009 saat Ryan Dahl memutuskan untuk menciptakan *asynchronous networking server.* Node.js sendiri diperkenalkan pertama kali ke publik pada JSConf di awal tahun 2009. Node.js menyediakan sebuah *platform* untuk pengembangan aplikasi jaringan yang menggunakan bahasa pemrograman JavaScript.

Node.js menyediakan solusi untuk beberapa isu terkait komunikasi *real-time* pada *web server* dan didukung oleh beberapa raksasa industri *web* seperti Google dan Microsoft. Dengan adanya dukungan dari raksasa industri *web*, maka pengembangan Node.js menjadi sangat cepat dan baik.

Dalam pengembangan sebuah *web* aplikasi, aspek keamanan harus menjadi prioritas. Sebuah *web* aplikasi memungkinkan untuk menyimpan serta menggunakan data-data atau informasi dari pengguna yang banyak termasuk dalam kategori yang rahasia. Oleh karena itu, *web* aplikasi perlu mempunyai mekanisme untuk melakukan pengamanan terhadap data/informasi yang rahasia. Salah satu istilah dalam keamanan web aplikasi adalah kerentanan (*vulnerability*).

Kerentanan (*vulnerability*) merupakan atribut atau karakteristik dari suatu komponen yang dapat dimanfaatkan oleh pihak eksternal atau internal untuk melanggar kebijakan keamanan atau menyebabkan kerusakan terhadap komponen itu sendiri, dan/atau sistem atau infrastruktur pada sistem tersebut.

Untuk mengetahui adanya kerentanan tersebut, maka dalam pengembangan aplikasi atau perangkat lunak perlu dilakukan *vulnerability assessment* (VA). Hal ini dilakukan untuk menghindari adaanya celah keamanan dalam aplikasi atau perangkat lunak tersebut. Potensi resiko keamanan *web* yang paling sering muncul pada aplikasi web menurut *Open Web Application Security Project* (OWASP) ditunjukkan pada Tabel 1-1 [11].

Tabel 1-1 Daftar Risiko Keamanan *Web* Aplikasi [11]

No.	Resiko	Deskripsi
1	*Infection*	*Injection flaws* bisa berupa injeksi SQL, OS, dan LDAP. Terjadi karena *untrusted data* dikirim ke sebuah *interpreter* yang akan memberikan perintah atau akses data yang *unauthorized*.
2	*Cross-Site Scripting (XSS)*	XSS terjadi karena sebuah aplikasi mengambil *untrusted data* dan mengirim data tersebut ke peramban tanpa melalui validasi. XSS memungkinkan penyerang mengeksekusi *scripts* pada peramban yang akan diserang dengan mengambil *user sessions*, *deface web sites*, atau *redirect* pengguna ke situs lainnya

3	*Broken Authentication and Session Management*	Fungsi aplikasi yang terkait dengan *authentication* dan *session management* seringkali tidak benar implementasinya, sehingga penyerang dapat mengambil kata sandi, kunci, *session tokens*, atau *exploit* kelemahan lainnya untuk mengambil data identitas pengguna.
4	*Insecure Direct Object References*	Akses ke objek referensi terjadi bila pengembang *exposes* sebuah referensi dari sebuah *internal implementation* seperti berkas, direktori, atau kunci basis data tanpa adanya pengecekan *access control* atau proteksi, sehingga penyerang memanipulasi referensi tersebut yang dapat digunakan untuk melakukan akses *unauthorized data.*
5	*Cross-Site Request Forgery (CSRF)*	Serangan CSRF terjadi karena seorang pengguna aplikasi yang melakukan *login* akan dialihkan ke aplikasi *web* yang sudah dipalsukan, kemudian data pengguna tersebut akan diambil.
6	*Security Misconfiguration*	Kesalahan konfigurasi keamanan terjadi karena adanya kesalahan konfigurasi pada aplikasi, *framework, server* aplikasi, *web server*, *server database,* dan *platform.*
7	*Insecure Cryptographic Storage*	Tidak adanya proteksi yang mencukupi terhadap data yang sensitif, misalnya kartu kredit, dan *authentication credentials,* sehingga memungkinkan terjadinya pencurian identitas, *credit card fraud*, atau kejahatan lainnya

8	*Failure to Restrict URL Access*	Terjadinya kegagalan proteksi terhadap URL situs web yang diakses karena tidak adanya pengecekan otorisasi, sehingga memungkinkan pemalsuan URL.
9	*Insufficient Transport Layer Protection*	Kurangnya perlindungan pada *Transport layer* sehingga memungkinkan terjadinya *penggunaan expired* atau *invalid certificates*, atau karena tidak menggunakannya dengan benar.
10	*Unvalidated Redirects and Forwards*	Terjadi pengalihan/*redirect* dan meneruskan/*forward* ke situs *web* lain karena tidak adanya validasi terhadap halaman yang dituju tersebut. Hal ini memungkinkan terjadinya pengalihan ke situs *phishing.*

Node.js sebagai salah satu bahasa pemrograman yang banyak digunakan untuk mengembangkan aplikasi berbasis *web* harus memperhatikan pula sisi keamanan pada aplikasi. Komunitas Node.js telah cukup banyak menyediakan modul-modul serta sejumlah *library* yang dapat digunakan untuk mengembangkan aplikasi. Modul-modul tersebut akan memudahkan *developer* untuk mengembangkan aplikasi berbasis *web*. Terdapat beberapa modul serta petunjuk yang dikembangkan oleh komunitas Node.js pada aspek keamanan aplikasi. Pada makalah ini, akan dibahas beberapa contoh teknik untuk menerapkan aspek keamanan pada aplikasi yang dikembangkan menggunakan Node.js.

Bab II Dasar Teori

2.1 Sejarah JavaScript

JavaScript dikembangkan oleh Brendan Eich di bawah nama Mocha yang merupakan peningkatan dari *browser* Netscape dan diubah namanya menjadi LiveScript pada *beta release* di tahun 1995. Kemudian diubah namanya menjadi JavaScript sebagai salah satu strategi *marketing* dan secara resmi diluncurkan oleh Netscape 2.0 pada Maret 1996. Microsoft mengimplementasikan versi tersendiri dari bahasa tersebut dengan nama JScript. Kedua bahasa pemrograman tersebut kemudian distandarisasi di bawah nama ECMA-262 (ECMAScript) dan kemudian *European Computer Manufacturers Association* (ECMA) menyetujui untuk nama JavaScript.

Pada *server side*, JavaScript pertama kali diimplementasikan oleh Netscape pada tahun 1996. Netscape merencanakan untuk mengenalkan *browser* yang murni menggunakan Java sehingga membutuhkan implementasi dari JavaScript, yang disebut proyek "Javagator". Walaupun kemudian proyek tersebut dihentikan, masih terdapat sub-proyek yang disebut "Rhino". Rhino kemudian dikembangkan menjadi proyek Mozilla.org (cikal bakal *Mozilla Foundation*) pada tahun 1998, dan terus dikembangkan. Proyek tersebut tidak didukung oleh *developer* pada umumnya sehingga proyek tersebut mempunyai progres yang lambat. Proyek ini pun kemudian terhenti pada ECMAScript 3 yang sudah tidak dikembangkan lagi.

Pada tahun 2006, Google mulai memasuki pasar *web browser* dan membutuhkan JavaScript *engine* untuk menjalankan *browser* versi Google tersendiri yang saat ini dikenal sebagai Google Chrome. Pengembangan *web browser* tersebut dimulai dengan sebuah JavaScript *engine* yang disebut V8. V8 melakukan kompilasi pada JavaScript dan menghasilkan kode dalam bahasa mesin, sehingga performa menjadi lebih baik. *Engine* tersebut dikembangkan agar bisa *embeddable*, sehingga dapat digunakan di seluruh *environment* yang mendukung C++. *Engine* V8 inilah yang kemudian digunakan oleh Ryan Dahl untuk mengembangkan Node.js.

JavaScript mengalami perkembangan yang masif tapi kacau karena sifatnya yang *multi-faced* sehingga diperlukan adopsi dan pembelajaran lebih lanjut untuk fitur yang baru. JavaScript dimulai sebagai sebuah bahasa *scripting* yang mudah untuk memberikan interaktivitas pada halaman statis HTML. Akan tetapi, saat ini JavaScript telah menjadi sebuah bahasa pemrograman yang kuat dan digunakan pada setiap *desktop*.

2.2 Node.js

Node.js adalah sebuah perangkat lunak yang di desain untuk mengembangkan aplikasi berbasis *web*. Node.js sendiri dieksekusi sebagai aplikasi *server*. *Platform* ini menggunakan bahasa pemrograman JavaScript dan menggunakan teknik *non-blocking* untuk mempercepat proses. Teknik *non-blocking* adalah sebuah metode penyelesaian sebuah fungsi.

Node.js sendiri diperkenalkan oleh Ryan Dahl pada tahun 2009 dan mengalami perkembangan yang pesat sejak diperkenalkan pada JSConf di awal tahun 2009. Sebelumnya, JavaScript merupakan bahasa pemrograman yang banyak digunakan untuk membuat aplikasi berbasis web pada sisi *client* , dan masih banyak digunakan hingga saat ini.

Pada umumnya, aplikasi *server* menggunakan bahasa PHP, Java, Ruby, maupun Python yang telah dikenal sebelumnya, sedangkan Bahasa JavaScript lebih umum digunakan untuk mengembangkan aplikasi *client* yang membuat aplikasi tersebut menjadi lebih interaktif. Pada dasarnya, Node.js merupakan JavaScript yang dieksekusi sebagai *server-side*. Salah satu kelebihan dari Node.js adalah adanya teknik *non-blocking*. Teknik *non-blocking* adalah teknik dengan Node.js akan melakukan eksekusi secara independen. Node.js akan mengeksekusi sebuah operasi tanpa harus menunggu operasi sebelumnya selesai dieksekusi, sehingga menghasilkan aplikasi *web* yang lebih cepat dan efisien. Node.js dengan teknik *non-blocking* ini dapat diilustrasikan sebagai sebuah restoran. Dalam industri restoran, proses pertama adalah pelayan akan mencatat pesanan dari konsumen kemudian menyerahkannya ke pada juru masak. Kemudian, pelayan tersebut akan

melayani konsumen lainnya tanpa harus menunggu pesanan sebelumnya selesai dimasak dan dihidangkan.

Aplikasi Node.js dapat ditemukan dan diunduh pada *website* resmi dari Node.js pada http://nodejs.org/.

2.3 Sejarah Node.js

Pada pengembangannya di awal tahun 2009, Ryan Dahl memutuskan untuk mengembangkan sebuah *networking server* yang mampu melakukan operasi *asynchronous* I/O. Ryan Dahl memutuskan untuk menggunakan V8 JavaScript *engine* yang telah dikembangkan oleh raksasa *web* aplikasi Google karena *engine* tersebut telah dioptimasi untuk menjalankan JavaScript pada lingkungan UNIX. Hal inilah yang merupakan cikal bakal dari Node.js.

Node.js pertama kali diperkenalkan ke publik pada JSConf pada tahun 2009. Pada saat itu, Ryan Dahl mengenalkan Node.js versi 0.1. Hal tersebut memperoleh atensi yang tinggi karena penemuan tersebut adalah hal yang baru dan menarik, serta menyediakan solusi dari beberapa isu dalam pengembangan *web* modern. Tidak seperti model lain yang telah banyak digunakan, Node.js mendukung *event model* dari tingkat *language*. Itulah mengapa Node.js menyediakan solusi yang baik untuk pengembangan layanan *web* yang *real-time*, yang banyak dibutuhkan pada pengembangan *web* aplikasi modern. Selain itu, Node.js menggunakan JavaScript sebagai bahasa pemrograman, dan JavaScript merupakan bahasa pemrograman yang telah banyak digunakan untuk mengembangkan aplikasi *web*, sehingga *developer* akan cukup familiar dan lebih mudah untuk menggunakan *platform* Node.js.

2.4 *Node Package Manager* (NPM)

Salah satu alasan Node.js menjadi sangat popular adalah adanya *Node Package Manager* (NPM). NPM akan memudahkan *developer* untuk membangun dan menggunakan modul-modul yang diperlukan sebagai pendukung Node.js. pada dasarnya NPM merupakan salinan dari *package manager* pada beberapa *platform*

Linux. Hal ini membuat Node.js mempunyai komunitas yang besar dan berkembang dengan pesat.

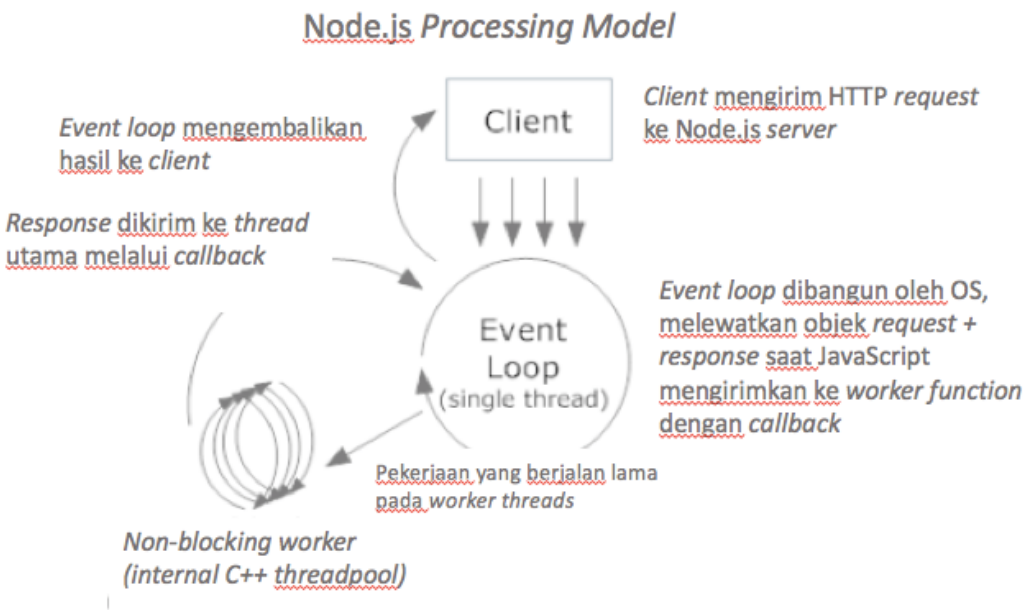
Node.js juga didukung oleh beberapa pemain besar pada pengembangan *web* aplikasi seperti Google dan Microsoft. Selain itu terdapat beberapa *web service* yang menggunakan Node.js sebagai *platform* dasar, seperti LinkedIn [7] dan Azure [16].

Node Package Manager (NPM) sendiri dikembangkan oleh komunitas penggiat Node.js dan terus mengalami revisi serta perubahan mengikuti kebutuhan para *developer* dan bertujuan pula untuk menutup celah-celah keamanan yang terdapat pada modul-modul Node.js di versi-versi sebelumnya. *Node Package Manager* dapat diunduh pada website https://www.npmjs.com .

Bab III PEMBAHASAN

3.1 Node.js *Processing Model*

Salah satu ide terbaru dari Node.js adalah pengguna dipaksa untuk menuliskan kode *non-blocking* karena *function* pada Node.js sendiri berupa *non-blocking*. Terdapat beberapa sistem yang lebih kompleks didalamnya, akan tetapi hampir seluruh dari *library* dibuat pada *blocking platform*. Node.js mendukung *event model* pada *language level*. Menjadi *asynchronous* dan *non-blocking* merupakan salah satu ide utama dari penciptaan Node.js [14]. Gambar berikut akan menjelaskan mengenai proses yang dilakukan Node.js.



Gambar 3-1 Node.js *Processing Model*

Langkah awal *client* mengirimkan HTTP *request* ke Node.js *server*. Kemudian Sistem Operasi akan membuat suatu *event loop* yang menangani sebuah *request* dari *client*. Setiap *event loop* yang dibangun tersebut akan diteruskan ke *non-blocking worker* yang dijalankan oleh *internal C++ threadpool*. *Non-blocking*

worker dapat bekerja secara *multithreading*. Proses tindak lanjut dari *request client* tidak lagi dijalankan oleh *event loop*, akan tetapi dijalankan oleh *non-blocking* worker yang dapat bekerja *multithreading*. Oleh karena itu, *request* dari *client* berikutnya dapat diterima oleh *server* dan dibangun *event loop* kembali tanpa harus menunggu tindak lanjut dari *request* sebelumnya. Setelah *request* dari *client* selesai di proses oleh *non-blocking worker*, maka *response* akan dikembalikan ke *thread* utama melalui *callback,* yang kemudian diteruskan ke *client* sebagai *response*.

3.2 Node.js *Security Checklist*

A. Manajemen Konfigurasi

1. Keamanan HTTP Headers

Terdapat beberapa kemanan terkait HTTP *Headers* yang harus diperhatikan saat mengembangkan sebuah *web server*. *Headers* tersebut adalah:

- **Strict-Transport-Security** yang digunakan untuk memaksa koneksi yang aman ke server melalui SSL/TLS.
- **X-Frame-Options** yang digunakan untuk proteksi *Clickjacking*.
- **X-XSS-Protection** yang mengaktifkan *filter Cross Site Scripting* (XSS).
- **X-Content-Type-Options** yang mencegah *browser* dari MIME-*sniffing* .
- **Content-Security-Policy** yang mencegah sejumlah serangan termasuk *Cross Site Scripting* dan *cross site injection* lainnya.

Pada NodeJ.js untuk memenuhi komponen HTTP *Headers* dapat menggunakan modul Helmet (https://www.npmjs.com/package/helmet) sebagai berikut:

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


var app = express();
```

```
app.use(helmet());
```

Di dalam banyak arsitektur *web* aplikasi, *header* dapat pula di atur pada konfigurasi *web server* (Apache, Nginx) tanpa harus mengubah pada kode aplikasi. Dalam Nginx, konfigurasi dapat berupa sebagai berikut:

```
# nginx.conf


add_header X-Frame-Options SAMEORIGIN;

add_header X-Content-Type-Options nosniff;

add_header X-XSS-Protection "1; mode=block";

add_header Content-Security-Policy "default-src 'self'";
```

Untuk melakukan pengecekan apakah *header* dari web aplikasi telah benar atau tidak, terdapat sebuah *website online checker* untuk melakukan pengecekan tersebut. *Online checker* dapat ditemukan pada http://cyh.herokuapp.com/cyh. Aplikasi *online* tersebut akan meminta URL dari *website* dan akan memberikan skor terkait HTTP *Headers*.

2. Data yang Sensitif pada Pihak *Client*

Saat melakukan *deployment* pada aplikasi *front end* , harus dipastikan bahwa rahasia dan *credential* dari API tidak terbuka pada *source code*, dan tidak dapat di baca oleh siapapun.

Untuk melakukan pengecekan pada *source code* berbasis Node.js, dapat dilakukan secara otomatis menggunakan *code review tool*, seperti :

- Mocha (http://mochajs.org/)
- Cucumber (https://cucumber.io/)
- Instanbul (https://github.com/gotwarlost/istanbul)
- Coverolls (https://coveralls.io/)
- JShint (https://github.com/jshint/jshint)

- Code Climate (https://codeclimate.com/)
- David (https://david-dm.org/)
- Libraries.io (https://libraries.io/npm)

Penggunaan lebih dari satu dan banyak *tool* lebih direkomendasikan karena dengan banyaknya *tools* yang digunakan, maka referensi dari *code review* akan semakin baik, sehingga meningkatkan kualitas dari kode yang dibentuk. Akan tetapi, tetap diperlukan peninjauan ulang kembali kode program yang telah dibuat secara manual. Hal ini untuk mencegah adanya kode-kode yang tidak dapat ditentukan kualitas serta keamanannya oleh *tool* yang otomatis.

B. Autentikasi

1. Proteksi *Brute-Force*

Brute-Force merupakan proses enumerasi yang sistematis untuk seluruh kandidat yang mungkin sebagai sebuah solusi dan melakukan pengecekan apakah kandidat tersebut memenuhi masalah yang dihadapi. Pada sebuah *web* aplikasi, sebuah isian untuk *login* dapat menjadi kandidat yang tepat.

Untuk melakukan proteksi pada aplikasi dari jenis serangan ini, Node.js mempunyai *package* yang disebut *ratelimiter*. Berikut merupakan contoh implementasi dari *ratelimiter package*:

```
var email = req.body.email;

var limit = new Limiter({ id: email, db: db });


limit.get(function(err, limit) {


});
```

Dalam proses proteksi tersebut, dapat juga digunakan *middleware* untuk Node.js seperti *Express* atau *Koa*. Dalam *Koa*, proteksi untuk *Brute-Force* dapat dilakukan seperti contoh berikut:

```
var ratelimit = require('koa-ratelimit');
var redis = require('redis');
var koa = require('koa');
var app = koa();

var emailBasedRatelimit = ratelimit({
  db: redis.createClient(),
  duration: 60000,
  max: 10,
  id: function (context) {
    return context.body.email;
  }
});

var ipBasedRatelimit = ratelimit({
  db: redis.createClient(),
  duration: 60000,
  max: 10,
  id: function (context) {
    return context.ip;
  }
});

app.post('/login', ipBasedRatelimit, emailBasedRatelimit,
handleLogin);
```

Yang dilakukan pada proses diatas adalah membatasi seberapa banyak percobaan yang boleh dilakukan dalam satu waktu. Hal ini dilakukan untuk mengurasi risiko dari serangan *brute-force*.

Untuk melakukan pengetesan terhadap serangan *brute-force*, dapat menggunakan *tools* seperti Hydra (terdapat pada Sistem Operasi Kali Linux).

C. *Session Management*

1. *Cookie Flags*

Terdapat beberapa *attribute* dari sebuah *cookie* yang digunakan untuk keamanan *cookie* tersebut, yaitu:

- **Secure** – *attribute* ini akan memberitahu *browser* untuk hanya mengirimkan *cookie* jika *request* dikirim melalui *channel* yang aman, yaitu HTTPS.
- H**ttpOnly** – *attribute* ini digunakan untuk mencegah serangan seperti *cross-site scripting*, karena dengan mengaktifkan *flag* ini, maka *cookie* tidak dapat diakses melalui JavaScript.

2. *Cookie Scope*

Terdapat beberapa komponen yang penting untuk sebuah *cookie*, antara lain:

- **Domain** – *attribute* ini digunakan untuk membandingkan antara domain *server* dan URL yang diminta. Jika domain cocok atau termasuk dalam sub-domain, maka kemudian *attribute* **path** akan melakukan pengecekan selanjutkan.
- **Path** – dalam sebuah domain, URL *path* dari *cookie* dapat lebih spesifik. Jika domain dan *path* telah cocok, maka *cookie* akan dikirim sesuai *request.*
- **Expires** – *attribute* ini digunakan untuk mengatur waktu *cookie* akan kedaluarsa, karena *cookie* akan tetap dapat digunakan apabila *cookie* tersebut belum kedaluarsa.

Dalam Node.js, untuk melakukan pengaturan *attribute* pada *cookie,* dapat menggunakan *cookie package*. Dapat digunakan *wrapper* untuk melakukan hal ini, seperti *cookie-session.*

```
var cookieSession = require('cookie-session');

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

```
var app = express();


app.use(cookieSession({
  name: 'session',
  keys: [
    process.env.COOKIE_KEY1,
    process.env.COOKIE_KEY2
  ]
}));


app.use(function (req, res, next) {
  var n = req.session.views || 0;
  req.session.views = n++;
  res.end(n + ' views');
});


app.listen(3000);
```

3. *Cross-Site Request Forgery*

Cross-Site Request Forgery (CSRF) adalah sebuah serangan yang memaksa pengguna untuk mengeksekusi aksi yang tidak diinginkan pada suatu aplikasi *web* saat pengguna sudah masuk ke *web* aplikasi (*login).*

Dalam pengembangan aplikasi menggunakan Node.js, terdapat modul yang dapat digunakan untuk mengantisipasi CSRF, yaitu *csrf module*. Modul tersebut merupakan modul *low-level* dari Node.js. Oleh karena itu, dapat digunakan *wrapper* untuk *framework* yang berbeda. Salah satu contoh modul csrf adalah *csurf*

module yang merupakan proteksi CSRF dari *express middleware.* Berikut merupakan contoh implementasi dari menggunakan modul *csurf.*

```
var cookieParser = require('cookie-parser');
var csrf = require('csurf');
var bodyParser = require('body-parser');
var express = require('express');

// setup route middlewares
var csrfProtection = csrf({ cookie: true });
var parseForm = bodyParser.urlencoded({ extended: false });

// create express app
var app = express();

// we need this because "cookie" is true in csrfProtection
app.use(cookieParser());

app.get('/form', csrfProtection, function(req, res) {
  // pass the csrfToken to the view
  res.render('send', { csrfToken: req.csrfToken() });
});

app.post('/process', parseForm, csrfProtection, function(req,
res) {
  res.send('data is being processed');
});
```

Pada *view layer*, akan dihasilkan CSRF *token* seperti berikut:

```
<form action="/process" method="POST">
```

```
  <input type="hidden" name="_csrf" value="{{csrfToken}}">


  Favorite color: <input type="text" name="favoriteColor">

  <button type="submit">Submit</button>

</form>
```

D. Validasi Data

1. *Cross-Site Scripting* (XSS)

Cross-Site Scripting merupakan teknik serangan dengan memasukkan kode JavaScript pada sebuah *web* aplikasi melalui *entry point* yang terdapat pada *web* aplikasi, seperti *open-form,* dan URL *parameter.* Terdapat dua jenis dari *Cross-Site Scripting*, yaitu *Reflected Cross-Site Scripting* dan *Stored Cross-Site Scripting.*

- *Reflected Cross-Site Scripting* terjadi ketika penyerang menginjeksi kode JavaScript yang *executable* ke dalam HTML *response* .
- *Stored Cross-Site Scripting* terjadi ketika aplikasi menyimpan masukan yang tidak di-*filter* dengan baik. Apabila penyerang memasukkan kode JavaScript yang *executable* dan kemudian kode tersebut disimpan dalam *database,* maka web aplikasi akan mengeksekusi kode tersebut terus-menerus apabila web aplikasi tersebut diakses oleh pengguna.

Untuk melakukan pertahanan pada jenis serangan ini, maka harus terdapat mekanisme sanitasi *input* agar *input* yang diberikan oleh pengguna bukan berupa kode JavaScript. Sanitasi *input* ini tergantung pada jenis masukan yang dibutuhkan.

2. SQL *Injection*

SQL *injection* merupakan injeksi dari SQL *query* yang parsial atau menyeluruh melalui masukan pengguna. SQL *injection* dapat membuat data yang sensitif dapat bocor dan dapat disalahgunakan.

Berikut merupakan contoh dari SQL *query* :

```
select title, author from books where id=$id
```

Pada contoh diatas, `$id` merupakan parameter yang dimasukkan oleh pengguna. Salah satu teknik yang paling umum digunakan dalam SQL *injection* adalah menggunakan perbandingan yang menghasilkan nilai *true*, sehingga seluruh data yang tersimpan dalam tabel *database* akan menjadi keluaran.

Pada kasus diatas, apabila parameter `$id` diisi dengan nilai `2 or 1=1`, maka *query* akan menjadi sebagai berikut:

```
select title, author from books where id=2 or 1=1
```

Pada *query* tersebut, terdapat parameter `1=1`, yang menghasilkan nilai *true,* sehingga *query* tersebut akan mengembalikan seluruh nilai yang terdapat pada kolom `title`, dan `author` dari tabel `books`. Hal ini akan sangat berbahaya karena seluruh data menjadi nilai keluaran, padahal nilai keluaran yang diinginkan adalah hanya `title, dan author` dari `id` yang spesifik.

Untuk mencegah atau menanggulangi serangan ini, maka diperlukan parameterisasi terhadap *query* serta diperlukan sanitasi *input* dari variabel yang digunakan sebelum nilai variabel tersebut dimasukkan ke dalam *query*.

Salah contoh untuk melakukan parameterisasi *query* pada Node.js yang mengggunakan PostgreSQL dapat menggunakan *node-postgres module*. Untuk melakuka parameterisasi *query* dapat dilakukan sebagai berikut:

```
var q = 'SELECT name FROM books WHERE id = $1';

client.query(q, ['3'], function(err, result) {});
```

Pengecekan (*testing*) diperlukan untuk memverifikasi *web* aplikasi telah dapat menangani SQL *injection* atau tidak. Salah satu *open source tool* yang dapat digunakan untuk melakukan SQL *injection* secara otomatis adalah **sqlmap**. Sqlmap merupakan sebuah aplikasi berbasis konsol pada Sistem Operasi Linux yang dapat mensimulasikan serangan SQL *injection* dan memberikan laporan yang sistematis.

3. *Command Injection*

Command Injection adalah sebuah teknik yang digunakan oleh penyerang untuk menjalankan perintah system operasi pada *remote web server.* Dengan pendekatan

ini, penyerang mungkin dapat memperoleh informasi – informasi penting dari system, seperti *username, password,* serta informasi lainnya.

Amati contoh berikut. Apabila terdapat sebuah URL seperti :

```
https://example.com/downloads?file=user1.txt
```

URL tersebut menunjukkan bahwa *file* harus berupa text dengan ektensi `.txt` . Akan tetapi, penyerang dapat mengubah nilai dari parameter `file` menjadi sebagai berikut:

```
https://example.com/downloads?file=%3Bcat%20/etc/passwd
```

Pada contoh diatas, penyerang mengubah nilai dari parameter `file` menjadi sebuah perintah system operasi Linux, yaitu `"cat /etc/passwd"`.

Untuk mencegah terjadinya *command injection* , maka diperlukan sanitasi *input* dari pengguna.

E. Transmisi Data yang Aman

1. Versi SSL/TLS, Algoritma, dan Panjang Kunci

Protokol HTTP merupakan protocol yang mengirimkan data dalam bentuk *clear-text* sehingga sangat rentan akan pencurian data. Oleh karena itu, harus digunakan metode pengamanan pada HTTP menggunkan SSL/TLS *tunnel* yang sering disebut dengan HTTPS. Berdasarkan standar PCI DSS untuk keamanan aplikasi *web, tunnel* yang paling aman untuk digunakan adalah TLS 1.1 dan TLS 1.2, sedangkan untuk SSL 2.0 dan SSL 3.0 telah tidak direkomendasikan untuk digunakan karena mengandung algoritma kriptografi yang lemah [12].

Pada sebuah *web server*, diperlukan pengetesan pada *tunnel* keamanan yang digunakan pada aspek berikut:

- *Cipher,* kunci, dan konfigurasi dari *tunnel*
- Validitas sertifikat

Untuk melakukan pengetesan tersebut, dapat digunakan *tools* "nmap" untuk mengecek versi, sertifikat, serta algoritma yang digunakan pada *web server*.

```
nmap --script ssl-cert,ssl-enum-ciphers -p 443,465,993,995
www.example.com
```

2. HTTP *Strict Transport Security* (HSTS)

Dalam manajemen konfigurasi sebelumnya, telah disinggung mengenai *Strict-Transport-Security Header* yang memaksa koneksi yang aman (HTTPS) dengan *server*. Berikut merupakan contoh dari sebuah domain :

```
strict-transport-security:max-age=631138519
```

Variabel `max-age` berisi nilai dalam satuan detik yang mengharuskan *browser* untuk secara otomatis mengubah seluruh HTTP *request* menjadi HTTPS.

BAB IV KESIMPULAN

Node.js merupakan sebuah *platform* untuk mengembangkan *web server* berbasis JavaScript. Node.js mempunyai keunggulan utama pada metode *non-blocking* yang membuat *request* dari *client* dapat dijalankan seluruhnya dalam satu waktu sehingga mempercepat proses pekerjaan yang dilakukan *web server*.

Node.js digunakan untuk mengembangkan aplikasi berbasis *web.* Terdapat beberapa kerentanan aplikasi berbasis *web* yang dijabarkan oleh OWASP. Terdapat beberapa *checklist* yang harus dipenuhi untuk membangun sebuah aplikasi berbasis *web* menggunakan Node.js sebagai pembangun *web server*, antara lain konfigurasi HTTP *Header* yang tepat, melakukan teknik pengamanan untuk autentikasi, melakukan pengamanan pada *session management*, serta beberapa *checklist* dari OWASP seperti pengamanan untuk serangan *Cross Site Request Forgery* (CSRF), *Cross Site Scripting* (XSS), *SQL Injection, Command Injection* , dan transmisi melalui *tunnel* yang aman.

Komunitas Node.js telah mengembangkan modul-modul yang dapat digunakan untuk mengembangkan *web server* berbasis Node.js. Terdapat banyak modul yang dapat digunakan untuk melakukan pengamanan terhadap aplikasi Node.js, sehingga pembuatan aplikasi Node.js yang aman dan efektif dapat dilakukan dengan lebih mudah.

DAFTAR PUSTAKA

[1] B. Carter, "HTML Educational Node.js System (HENS): An Applied System for Web Development," *Information and Computer Technology (GOCICT), 2014 Annual Global Online Conference on*, Louisville, KY, 2014, pp. 27-31.

[2] Express, "Express - node web framework," [Online] Available: http://expressjs.com/. [Diakses April 2016].

[3] J. Karro and Jie Wang, "Protecting Web servers from security holes in server-side includes," *Computer Security Applications Conference, 1998. Proceedings. 14th Annual*, Phoenix, AZ, 1998, pp. 103-111.

[4] K. Haizhou, "A Research of Attacking Access Controls Algorithms of Web Application," *Industrial Control and Electronics Engineering (ICICEE), 2012 International Conference on*, Xi'an, 2012, pp. 679-681.

[5] K. Lei, Y. Ma and Z. Tan, "Performance Comparison and Evaluation of Web Development Technologies in PHP, Python, and Node.js," *Computational Science and Engineering (CSE), 2014 IEEE 17th International Conference on*, Chengdu, 2014, pp. 661-668.

[6] L. Gui-hong, Z. Hua and L. Gui-zhi, "Building a Secure Web Server Based on OpenSSL and Apache," *E-Business and E-Government (ICEE), 2010 International Conference on*, Guangzhou, 2010, pp. 1307-1310. doi: 10.1109/ICEE.2010.334.

[7] LinkedIn, "LinkedIn Mobile | LinkedIn," [Online] Available: http://www.linkedin.com/static?key=mobile. [Diakses April 2016] .

[8] N. Mendes, A. A. Neto, J. Durães, M. Vieira and H. Madeira, "Assessing and Comparing Security of Web Servers," *Dependable Computing, 2008. PRDC '08. 14th IEEE Pacific Rim International Symposium on*, Taipei, 2008, pp. 313-322.

[9] Node.js, “Node.js,” [Online] Available : http://nodejs.org/. [Diakses Maret 2016].

[10] Ojamaa and K. Düüna, "Assessing the security of Node.js platform," *Internet Technology And Secured Transactions, 2012 International Conference for*, London, 2012, pp. 348-355.

[11] OWASP, “Top 10-2013-Top 10,” [Online]. Available: https://www.owasp.org/index.php/Top_10_2013-Top_10. [Diakses Mei 2016].

[12] PCI Council, “Information Supplement Migrating from SSL and Early TLS”. April 2015.

[13] R. Vibhandik and A. K. Bose, "Vulnerability assessment of web applications - a testing approach," *e-Technologies and Networks for Development (ICeND),2015 Forth International Conference on*, Lodz, 2015, pp. 1-6.

[14] S. T. a. S. Vinoski, “Node.js: Using JavaScript to Build High-Performance Network Programs,” *Internet Computing*, vol. 14, no. 6, pp. 80-83, 2010.

[15] Sheng-Kang Lin, "From Web server security to Web components security," *Security Technology, 2003. Proceedings. IEEE 37th Annual 2003 International Carnahan Conference on*, 2003, pp. 176-182.

[16] Windows Azure, “Windows Azure: Cloud Computing | Cloud Services | Cloud Application Development,” [Online] Available: http://www.windowsazure.com/en-us/. [Diakses Mei 2016].