* Crosss site scripting(XSS)
  + Hackers inject client-side scripts into viewed page -leading to dat leaks
  + Re-direct users to malicious sites.
  + XSS attacks occurs on input forms when not properly encoded or validated.

Cross Site forgery attack (CSRF)

* + It attacks the changes in applications state request.
  + Hackers send users link through chat or email and ask to change email address or transfer funds

Brute force attacks:

* + Hackers rely on automated softwares
  + It can crack encrypted passwords and PINS
  + Using default cookies name is threat and hackers can easily identify them.

NPM and potential security risk

Denial of service attacks

SQL injections

* + Bypass authentication

SELECT \* FROM Users WHERE userID = 100 OR 1=1;

Since 1=1 is always true, the above statement will return all the rows of the Users table. This can be potentially dangerous if your table contains information like usernames and passwords.

SELECT \* FROM Users WHERE Name ="" or ""="" AND Pass ="" or ""=""

The above example returns all the rows from the Users table and then deletes the Orders table. If your user enters something like 100; DROP TABLE Orders; for their userID, the SQL statement becomes:

SELECT \* FROM Users WHERE userID = 105; DROP TABLE Orders;

* Add or remove HTTP headers :
* That’s where **Helmet.js** comes in, a Node.js module for securing HTTP headers. It has 12 Node modules that interfere with Express, and each module provides configuration options for securing different HTTP headers.

Password Encryption in Node.js

Encryption has the following two types:

* **Symmetric:** This uses the same key for encryption and decryption
* **Asymmetric:** This uses different keys for encryption and decryption

Password and hashing :

You can use the library **bcryptjs** to secure passwords in Node.js. bcryptjs takes the password and salt, which is the number of times it should execute the hashing algorithm.

The example below uses different function calls for generating the hash and salt.

bcrypt.genSalt(saltRounds, function(err, salt) {  
 bcrypt.hash(myPlaintextPassword, salt, function(err, hash) {  
 // Store hashed password in database  
 });  
});

Or you could auto-generate a hash and salt using:

bcrypt.hash(PlainTextPassword, saltRounds, function(err, hash) {  
 // Store hashed password in database  
});

You can also compare your password to the hashed password stored in the database using the compare method. If the passwords match, compare returns true; if not, false.

bcrypt.compare(PlainTextPassword, hash, function(err, result) {  
 // result == true  
});

Dynamic Typing - Node JS

Protect against race condition :

* Use call backs to overcome race conditions
* function generateCsrf(req, res, next) {  
    
   //safe default!  
   req.session.csrfToken = null;  
    
   //generate a new token  
   var crypto = require('crypto');  
   crypto.randomBytes(48, function(err, buffer) {  
    
   var token = buffer.toString('hex');  
    
   //save the token in the session and proceed  
   req.session.csrfToken = token;  
    
   //let's try to print the token for this demo  
   console.log(token)  
    
   next();  
    
   });  
  };
* Authentication
* Access Control
* Safe File Handling
* Typecasting
* Tools for scanning Node.js apps (Acutinex, NodeJsScan, etc.)