A6-Sensitive Data Exposure

Exploitability: DIFFICULT | Prevalence: COMMON | Detectability: AVERAGE | Technical Impact: SEVERE

Description

This vulnerability allows an attacker to access sensitive data such as credit cards, tax IDs, authentication credentials, etc to conduct credit card fraud, identity theft, or other crimes. Losing such data can cause severe business impact and damage to the reputation. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.

Attack Mechanics

If a site doesn't use SSL/TLS for all authenticated pages, an attacker can monitor network traffic (such as on open wireless network), and steals user's session cookie. Attacker can then replay this cookie and hijacks the user's session, accessing the user's private data.

If an attacker gets access the application database, he or she can steal the sensitive information not encrypted, or encrypted with weak encryption algorithm

How Do I Prevent It?

- Use Secure HTTPS network protocol
- Encrypt all sensitive data at rest and in transit
- Don't store sensitive data unnecessarily. Discard it as soon as possible.
- Ensure strong standard algorithms and strong keys are used, and proper key management is in place.
- Disable autocomplete on forms collecting sensitive data and disable caching for pages that contain sensitive data.

Source Code Example

1.The insecure demo application uses HTTP connection to communicate with server. A secure HTTPS sever can be set using https module. This would need a private key and certificate. Here are source code examples from `/server.js`

```
// Load keys for establishing secure HTTPS connection
var fs = require("fs");
var https = require("https");
var path = require("path");
var httpsOptions = {
    key: fs.readFileSync(path.resolve(__dirname, "./app/cert/key.pem")),
    cert: fs.readFileSync(path.resolve(__dirname, "./app/cert/cert.pem"))
};
```

2. Start secure HTTPS sever

```
// Start secure HTTPS server
https.createServer(httpsOptions, app).listen(config.port, function() {
    console.log("Express https server listening on port " + config.port);
});
```

3. The insecure demo application stores users personal sensitive information in plain text. To fix it, The `data/profile-dao.js` can be modified to use crypto module to encrypt and decrypt sensitive information as below:

```
// Include crypto module
var crypto = require("crypto");

//Set keys config object
var config = {
    cryptoKey: "a_secure_key_for_crypto_here",
    cryptoAlgo: "aes256", // or other secure encryption algo here
    iv: ""
};

// Helper method create initialization vector
// By default the initialization vector is not secure enough, so we create our own
var createIV = function() {
    // create a random salt for the PBKDF2 function - 16 bytes is the minimum length acco
rding to NIST
    var salt = crypto.randomBytes(16);
    return crypto.pbkdf2Sync(config.cryptoKey, salt, 100000, 512, "sha512");
};

// Helper methods to encryt / decrypt
var encrypt = function(toEncrypt) {
    config.iv = createIV();
    var cipher = crypto.createCipheriv(config.cryptoAlgo, config.cryptoKey, config.iv);
    return cipher.update(toEncrypt, "utf8", "hex") + cipher.final("hex");
};

var decrypt = function(toDecrypt) {
    var decipher = crypto.createDecipheriv(config.cryptoAlgo, config.cryptoKey, config.i
v);
    return decipher.update(toDecrypt, "hex", "utf8") + decipher.final("utf8");
};

// Encrypt values before saving in database
user.ssn = encrypt(ssn);
user.dob = encrypt(dob);

// Decrypt values to show on view
user.ssn = decrypt(user.ssn);
user.dob = decrypt(user.dob);
```