Node Security

Group Research

Seraphine, Francisca, Hasanin and Raj

Code Your Future

July 21, 2018

Outline

- https (Raj)
- same origin (Hasanin qasem)
- cookies (Seraphine)
- xss (Francisca)

Recently, Nodejs has become a popular choice to develop backe-end applications as many other server-side languages.

Equally, many factos should be taken into account when it comes to secuering the applications.

- ✓ JavaScript as a language,
- ✓ Node.js and its API (event loop and error handling),
- ✓ Third-party modules interactign with the application and
- ✓ HTTP request(CSRF) and response (XSS) cycles of the applications.

Example

eval(alert(body.a);a=”10” $->$ 10
eval(”alert(‘Your query string was $document.location.search‘)”);

Example

avoid using var; use let, const and run in strict mode

Example

Object property descriptors

- ✓ writable : false read only
- ✓ enumerable :false not come up during for in loops
- ✓ configurable : false cannot be deleted

Node

JavaScript has exceptions built into the language as an error-handling construct. When an exception is thrown, there needs to be some code to detect that error and handle it appropriately. In the browser, an uncaught exception immediately halts any execution that takes place. This will not cause your web page to crash, but it has the potential to leave your application in an unstable place.
In Node.js, an uncaught exception will terminate the application thread. This is very different from other server-side programming language.
(Callback errors, EventEmitter error and handling Uncaught exceptions using process.on())

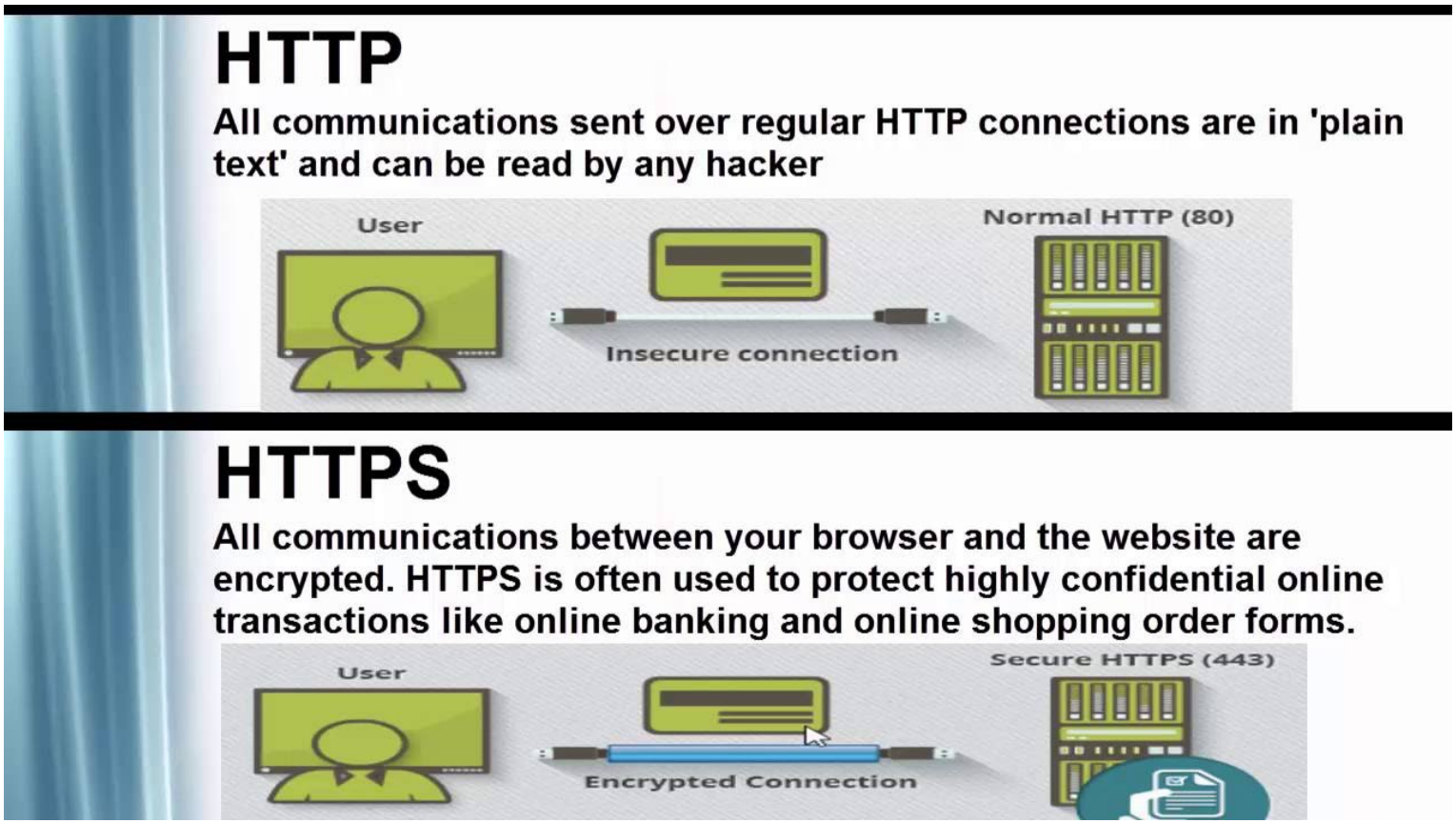
npm

You will take efforts to audit your code for security practices and you should also take an active role in monitoring the npm modules; you end up including in your project.

model

HTTP is the foundatation of data communication over the web. It works on request and response cycle. It is a stateless protocol: server does not have to retain information and status of each user for the duration of multiple requests. This can be acheived using cookies and sessions variables

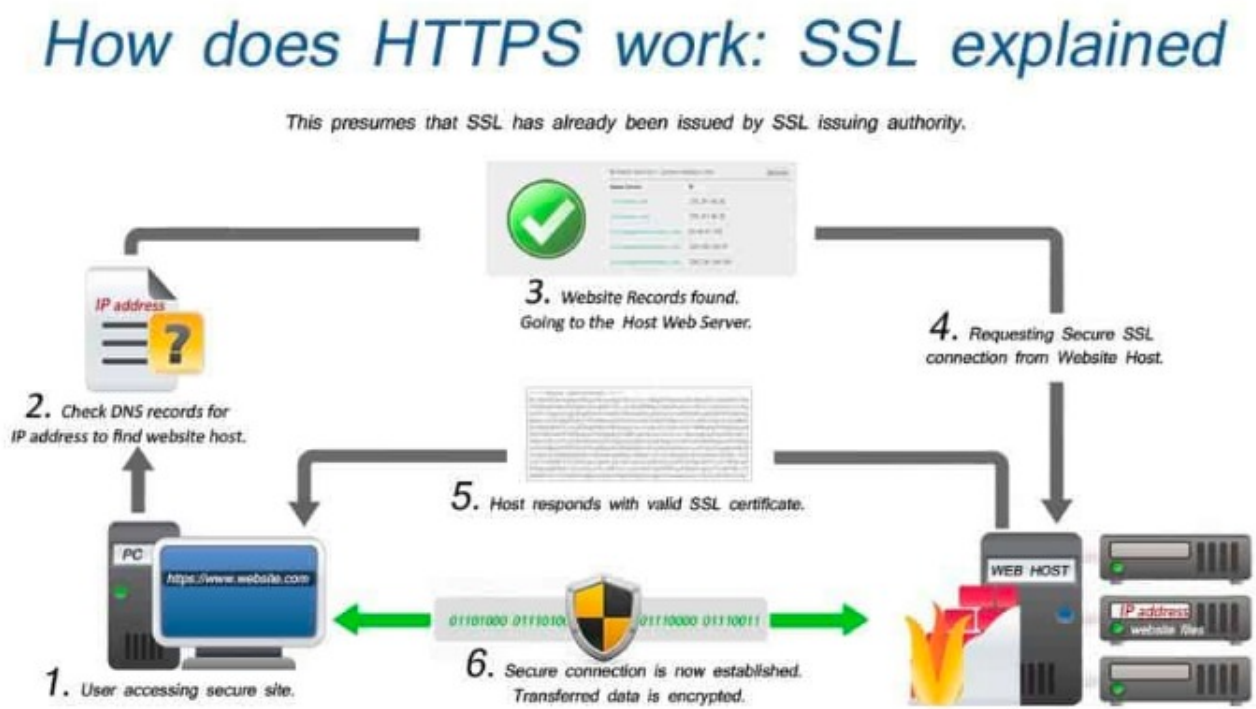
model


HTTP
All communications sent over regular HTTP connections are in 'plain text' and can be read by any hacker
User
Insecure connection
Normal HTTP (80)
HTTPS
All communications between your browser and the website are encrypted. HTTPS is often used to protect highly confidential online transactions like online banking and online shopping order forms.
User
Encrypted Connection
Secure HTTPS (443)

definition

HTTS is secure protocole with added securtity to the HTTP. Before data was sent and received as a plain text but now the data is encrypted. This provides addded security when communicating via HTPS. However, it does not mean the wesite is completely secure. We still have other attacks.

how it works?


How does HTTPS work: SSL explained
This presumes that SSL has already been issued by SSL issuing authority.
IP address
2. Check DNS records for IP address to find website host.
3. Website Records found. Going to the Host Web Server.
4. Requesting Secure SSL connection from Website Host.
5. Host responds with valid SSL certificate.
PC
https://www.website.com
1. User accessing secure site.
6. Secure connection is now established. Transferred data is encrypted.
WEB HOST
IP address
website files


https://accounts.google.com/Se

same origin

covered...

Hasanin

cookies

covered...

Seraphine

XSS

Francisca: contents

- What's XSS?
- XSS vulnerabilities
- XSS Goal and Types of XSS
- Usual types of attacks
- Methods of prevent XSS
- XSS Quiz

What's XSS?

Cross Site Scripting, otherwise known as XSS is a code injection attack allowing the injection of malicious code into a website. XSS is currently one of the most common website attacks, with almost every website requiring the user to have JavaScript turned on. Rather than being an attack on the website itself, it uses the website as a means to attack the users of that website. When you can get your XSS permanently on a website all those who visit that page will have the JavaScript executed by their browser.

XSS vulnerabilities

XSS vulnerabilities are especially dangerous because an attacker exploiting an XSS attack can gain the ability to do whatever the user can do, and to see what the user sees including passwords, payment and financial information, and more. Worse the victims, both the user and the vulnerable application, often wont be aware theyre being attacked.

XSS vulnerabilities...

The possibility of JavaScript being malicious becomes more clear when you consider the following facts:

- JavaScript has access to some of the user's sensitive information, such as cookies
- JavaScript can send HTTP requests with arbitrary content to arbitrary destinations by using XMLHttpRequest and other mechanisms
- JavaScript can make arbitrary modifications to the HTML of the current page by using DOM manipulation methods

Goal and Types of XSS

While the goal of an XSS attack is always to execute malicious JavaScript in the victim's browser, there are few fundamentally different ways of achieving that goal.
XSS attacks are often divided into five types:

a) JavaScript has access to some of the user's sensitive information, such as cookies

b) Ja Reflected XSS or Non-persistent XSS, which involves reflecting malicious script into a link on a page, The attack will activate once the link has been clicked, where the malicious string originates from the victim's request

c) DOM-based XSS, where the vulnerability is in the client-side code rather than the server-side code

Goal and Types of XSS...

d) Self-XSS, is a form of XSS vulnerability which relies on Social Engineering in order to trick the victim into executing malicious JavaScript code into their browser. Although it is technically not a true XSS vulnerability due to the fact it relies on socially engineering a user into executing code rather than a flaw in the affected website allowing an attacker to do so, it still poses the same risks as a regular XSS vulnerability if properly executed.

e) Mutated XSS (mXSS) XSS happens, when the attacker injects something that is seemingly safe, but rewritten and modified by the browser, while parsing the markup. This makes it extremely hard to detect or sanitize within the websites application logic. An example is rebalancing unclosed quotation marks or even adding quotation marks to unquoted

Usual types of attacks

Among many other things, the ability to execute arbitrary JavaScript in another user's browser allows an attacker to perform the following types of attacks:
a) Cookie theft The attacker can access the victim's cookies associated with the website using document.cookie, send them to his own server, and use them to extract sensitive information like session IDs.
b) Keylogging The attacker can register a keyboard event listener using addEventListener and then send all of the user's keystrokes to his own server, potentially recording sensitive information such as passwords and credit card numbers.

Usual types of attacks...

c) Phishing The attacker can insert a fake login form into the page using DOM manipulation, set the form's action attribute to target his own server, and then trick the user into submitting sensitive information.
Although these attacks differ significantly, they all have one crucial similarity: because the attacker has injected code into a page served by the website, the malicious JavaScript is executed in the context of that website. This means that it is treated like any other script from that website: it has access to the victim's data for that website (such as cookies) and the host name shown in the URL bar will be that of the website. For all intents and purposes, the script is considered a legitimate part of the website, allowing it to do anything that the actual website can.

Methods of prevent XSS

Recall that an XSS attack is a type of code injection: user input is mistakenly interpreted as malicious program code. In order to prevent this type of code injection, secure input handling is needed. In order to be truly vigilant against XSS and other common, debilitating vulnerabilities, like the rest of the OWASP Top 10, its important to use a mix of code review, automated static testing during development and dynamic testing once the application is live, in addition, of course, to using secure coding practices that will help prevent vulnerabilities like cross-site scripting.

Methods of prevent XSS...

For a web developer, there are three fundamentally different ways:
a) Escaping
The first method you can and should use to prevent XSS vulnerabilities from appearing in your applications is by escaping user input. Escaping data means taking the data an application has received and ensuring its secure before rendering it for the end user. By escaping user input, key characters in the data received by a web page will be prevented from being interpreted in any malicious way. In essence, youre censoring the data your web page receives in a way that will disallow the characters especially $<$ *and* $>$ characters from being rendered, which otherwise could cause harm to the application and/or users.

Methods of prevent XSS...

b) Validating Input
Expect any untrusted data to be malicious. Whats untrusted data? Anything that originates from outside the system and you dont have absolute control over so that includes form data, query strings, cookies, other request headers, data from other systems (i.e. from web services) and basically anything that you cant be 100% confident doesnt contain evil things.
Validating input is the process of ensuring an application is rendering the correct data and preventing malicious data from doing harm to the site, database, and users. While whitelisting and input validation are more commonly associated with SQL injection, they can also be used as an additional method of prevention for XSS.

Methods of prevent XSS...

c) Sanitizing

A third way to prevent cross-site scripting attacks is to sanitize user input. Sanitizing data is a strong defense, but should not be used alone to battle XSS attacks. Its totally possible youll find the need to use all three methods of prevention in working towards a more secure application. Sanitizing user input is especially helpful on sites that allow HTML markup, to ensure data received can do no harm to users as well as your database by scrubbing the data clean of potentially harmful markup, changing unacceptable user input to an acceptable format.

References

1. *https* : *//www.youtube.com/watch?v* = *M_nIIcKTxGk*
2. *https* : *//www.checkmarx.com/*2017/10/09/3 − *ways* − *prevent* − *xss/https* : *//excess* − *xss.com/*
3. *https* : *//www.incapsula.com/web* − *application* − *security/cross* − *site* − *scripting* − *xss* − *attacks.html*
4. *https* : *//en.wikipedia.org/wiki/Cross* − *site_scripting*
5. *https* : *//www.youtube.com/watch?v* = *po*3*zYOe*00*O*4

Conclusion

we have looked at basic security issues...

- learned to work as a group,
- learned to organize and manage a group work using GitHub and research to collect informtion and
- have created some level of security awareness while developing web applitcation using Nodejs