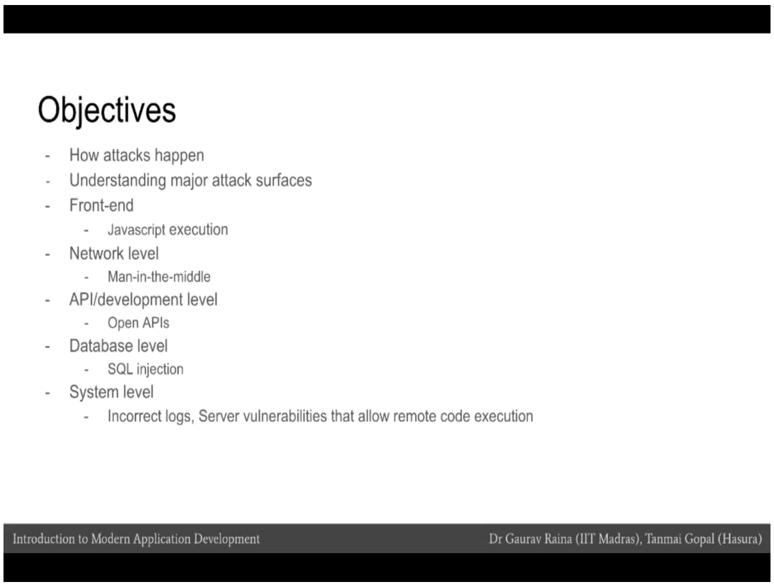
Introduction to Morden Application Development

Dr. Gaurav Raina
Prof. Tanmai Gopal
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Module – 14
Lecture – 25
Understanding security, and some best practices

(Refer Slide Time: 00:05)



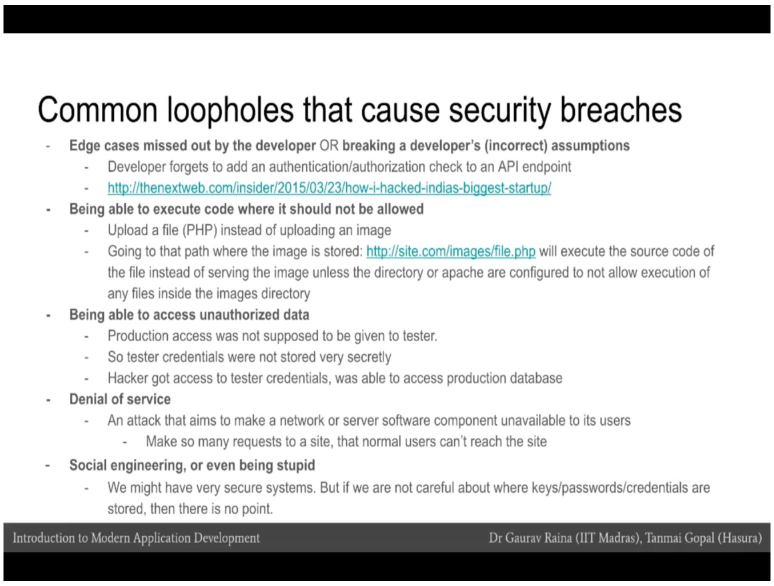
Objectives

- How attacks happen
- Understanding major attack surfaces
- Front-end
    - Javascript execution
- Network level
    - Man-in-the-middle
- API/development level
    - Open APIs
- Database level
    - SQL injection
- System level
    - Incorrect logs, Server vulnerabilities that allow remote code execution

Introduction to Modern Application Development   Dr Gaurav Raina (IIT Madras), Tanmai Gopal (Hasura)

Hi all, welcome to module 14. In this module we look at understanding the basic concepts at effect a security for a web app and looking at some common best practices.

(Refer Slide Time: 00:15)



Common loopholes that cause security breaches

- **Edge cases missed out by the developer** OR **breaking a developer's (incorrect) assumptions**
  - Developer forgets to add an authentication/authorization check to an API endpoint
  - http://thenextweb.com/insider/2015/03/23/how-i-hacked-indias-biggest-startup/
- **Being able to execute code where it should not be allowed**
  - Upload a file (PHP) instead of uploading an image
  - Going to that path where the image is stored: http://site.com/images/file.php will execute the source code of the file instead of serving the image unless the directory or apache are configured to not allow execution of any files inside the images directory
- **Being able to access unauthorized data**
  - Production access was not supposed to be given to tester.
  - So tester credentials were not stored very secretly
  - Hacker got access to tester credentials, was able to access production database
- **Denial of service**
  - An attack that aims to make a network or server software component unavailable to its users
    - Make so many requests to a site, that normal users can't reach the site
- **Social engineering, or even being stupid**
  - We might have very secure systems. But if we are not careful about where keys/passwords/credentials are stored, then there is no point.

Introduction to Modern Application Development — Dr Gaurav Raina (IIT Madras), Tanmai Gopal (Hasura)

This model is unfortunately going to be listing down quite a few point after points. So, it might get little boring, so bare with me. Let us look at the common loop holes that cause security breaches. The first most common loophole is edge cases that are missed out by the developer or in case the web app is being used by users on hackers in a way that breaks the developer's assumptions. The most common example of this is when the developer forgets to add an authentication or an authorization check to an API endpoint.

So; that means, that let us say you taking any API endpoint that places an order and you forget to check if the user id is locked in high profile example of this case was when Ola app was hacked by developer and the developers able to place recharges of an arbitrary amount of money. The second kind of example is when hackers are able to execute code where developers do not expect code to be executed. A very common example of this used to be when a amateur developers would setup a PHP service and would allow file uploads to happen. So, users would upload a file, but instead of say uploading an image which is for the developer who expecting user of the hacker upload in PHP file.

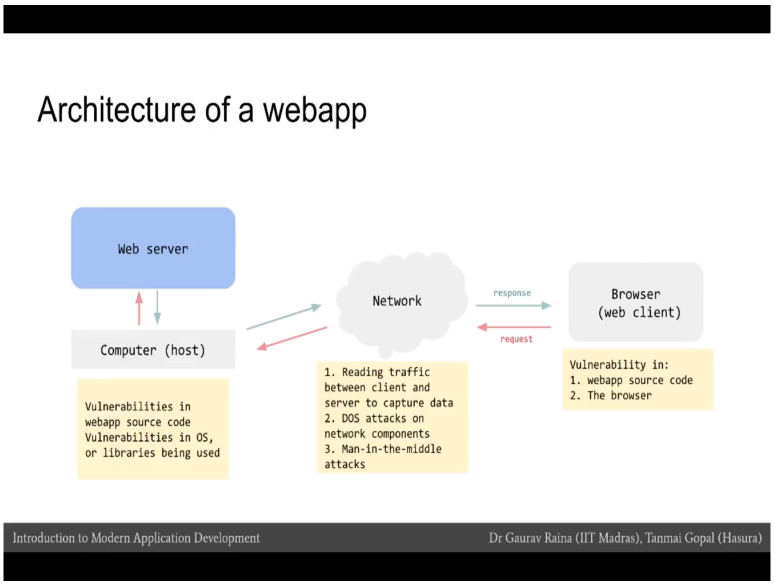
Now, if we visit this PHP file by trying to go to see the images folder and sets with file dot PHP, if the web server or the directory is not configured correctly, the PHP file will actually end up getting executed. So, this means that a random user who is not a developer is able to execute code on our server machine. So, the user or the hacker can potentially get access to all the different files that are there on the file system can even start making request to the database can try to find out of the database username and password is and such thing become possible.

So, another common loophole is when people who are not authorized enough are able to access data. So, a common example of this is when we have an application and we have a testing environment and a production environment. So, a testing environment is when we have an application that is deployed and tests server, and production environment is when its finally, a deployed on the actual server that is the app that are users use. So, the domain points to the production server and when we test the application we test to the test server. So, common every that happens in this case is that a developer or a tester who is testing things is mistakenly given credentials to access the production database.

Now, the tester might not know this, now the tester himself or herself might only malicious, but if the tester does not store credentials carefully then another user or hacker can get access to these credentials and they will be able to access the production database. The fourth kind of loophole which is often also at the control of developers or the organization is a denial of service attack. So, denial of service attack is a kind of an attack by hackers that aims to make either a part of a network or the server software on available to users and the typical way of doing this is that hackers would make so many requests to add site that normal users be not able to reach the side because the site is overloaded.

The last kind of example which is getting increasingly popular today is social engineering. So, the example we take all the best practices to secure our system, but we not careful about where we stored our credentials or password or keys.

(Refer Slide Time: 03:37)


Architecture of a webapp

Web server
Computer (host)
Vulnerabilities in webapp source code Vulnerabilities in OS, or libraries being used
Network
1. Reading traffic between client and server to capture data
2. DOS attacks on network components
3. Man-in-the-middle attacks
response
request
Browser (web client)
Vulnerability in:
1. webapp source code
2. The browser

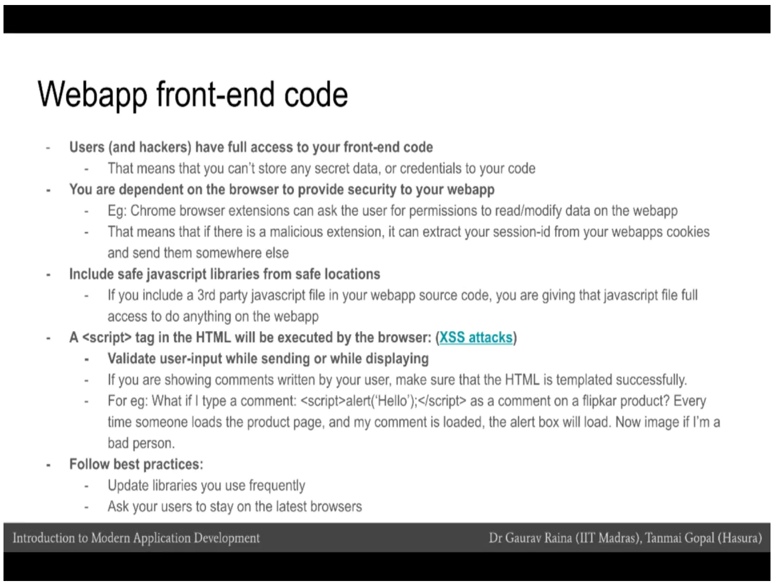
Introduction to Modern Application Development Dr Gaurav Raina (IIT Madras), Tanmai Gopal (Hasura)

So, these kinds of things are not really technical problem, but a just a problem with the way we manage or organized policies in our team. So, to understand what the sources of these different security vulnerabilities are, let us look at the architecture of a webapp. We will start from the right the browser is making a request to the network to the computer which is the server host and then to the web server software.

So, the first place where things in go wrong is browser itself, the vulnerability might be in the web app source code the code that we have written or might be in the browser itself. On the network it could be because hackers have access to the data that is in transit between a browser and a computer and their able to access that, it could be because there is a denial of service attack on network components which is not allowing users to access the server itself because also be because of man in the middle attacks where the network is being completely compromise and there is a hacker another agent that is sitting in the middle, that is pretending to be a web server and is not really a web server.

On the computer or on the server side of things they might be vulnerabilities in the web app source code or they might be vulnerabilities in the operating system or the library that is been used. These are three main touch points from where the main security vulnerabilities can come from and our objective is to try to understand what are the common things that go wrong and how we can prevent those common things from going wrong when we think about security.

(Refer Slide Time: 04:51)



Webapp front-end code

- **Users (and hackers) have full access to your front-end code**
    - That means that you can't store any secret data, or credentials to your code
- **You are dependent on the browser to provide security to your webapp**
    - Eg: Chrome browser extensions can ask the user for permissions to read/modify data on the webapp
    - That means that if there is a malicious extension, it can extract your session-id from your webapps cookies and send them somewhere else
- **Include safe javascript libraries from safe locations**
    - If you include a 3rd party javascript file in your webapp source code, you are giving that javascript file full access to do anything on the webapp
- **A <script> tag in the HTML will be executed by the browser: (XSS attacks)**
    - **Validate user-input while sending or while displaying**
    - If you are showing comments written by your user, make sure that the HTML is templated successfully.
    - For eg: What if I type a comment: <script>alert('Hello');</script> as a comment on a flipkar product? Every time someone loads the product page, and my comment is loaded, the alert box will load. Now image if I'm a bad person.
- **Follow best practices:**
    - Update libraries you use frequently
    - Ask your users to stay on the latest browsers

Introduction to Modern Application Development Dr Gaurav Raina (IIT Madras), Tanmai Gopal (Hasura)

So, first let us talk about the front end code and let us make some fundamental concepts clear, so that we understand where security vulnerabilities can come from. The first thing to understand is that on the browser users and hence hackers have full access to your front end code right the HTML the CSS and the javascript is completely accessible to any user who comes to our site this means that we cannot store any secret data or credentials or access token that a secret to our application in that source code. The second thing to understand is that we are dependent on the browser to provide security to our webapp.

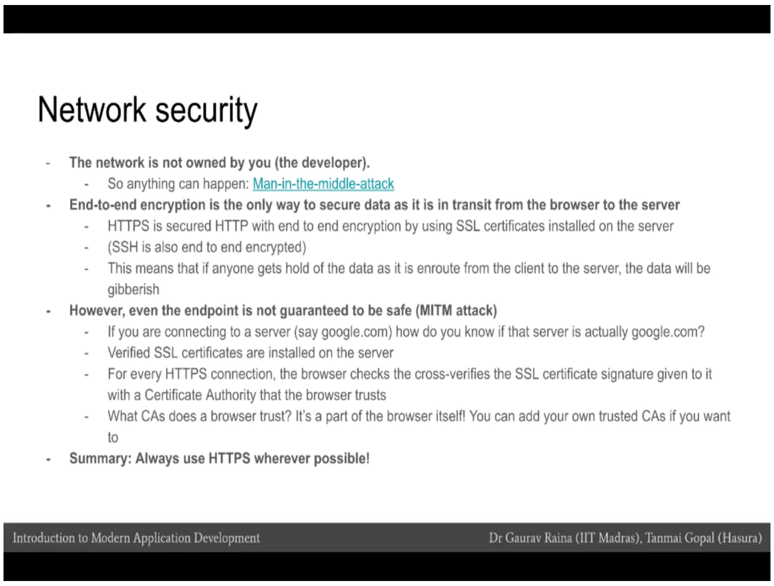
So, our application or entire application is only as safe as the browser is from the front end point of view. Just to give you an example of this when you use chrome browser extensions some browser extensions can ask the user for permissions to read and modify data on the webapp itself; that means, that if hypothetically there is a malicious extension or a bad chrome extension on the chrome app store, then and be installed this chrome extension then this chrome extension can extract a session id from a web apps cookies or can send them to some agent, and that agent can start using a session ids for making equation of a half.

Third thing that is important to remember is that if we include javascript libraries from third party locations, than those pieces of javascript code have full access to do anything on the webapp right because vs developer specified that those javascript file should be used; that means, that those javascript files have full access to do anything that they want on the web app. For example, if we are including say jquery javascript library and we loaded the jquery javascript file from an untrusted source. So, the un trusted source have might have modified the jquery library to have malicious species of code that are extracting user data the forth and point to remember is that if an our HTML for any reason our script tag is added either added by modifying HTML or added when the HTML was first loaded that code excite the script tag will be executed.

So; that means, for example, let say we have a common space and users are log type commons in each of the articles. So, let say I go to a comment box and I type javascript piece of code which is script alert hello and close the script tag; that means, if there is anybody else load this product page and the load the comment that I have tighten this script tag will execute and a hello we will show up in the alert box. Now imagine if I am a evil person and I do not write alert hello, but I right, but I write javascript that the user cannot see right in the javascript just starts executing.

So, it is important to validate the user input while users are sending that data or while displaying data that users have (Refer Time: 07:31) so; that means, the only way to sort of prevent this error is to ensure that if your web page is displaying content that in other user has written, it must be displayed in a way that the script tag is not included as if it HTML into the HTML document. A very simple thing that a lot of developers forget to do which become a source of common error and security risk is if we do not update library is that we used frequently for example, of using jquery we should make sure that we run the latest stable version of jquery and we should always also make sure that we keep asking a users to upgrade and stay on the latest browsers that do not have security vulnerabilities.

(Refer Slide Time: 08:10)



Network security

- The network is not owned by you (the developer).
    - So anything can happen: Man-in-the-middle-attack
- End-to-end encryption is the only way to secure data as it is in transit from the browser to the server
    - HTTPS is secured HTTP with end to end encryption by using SSL certificates installed on the server
    - (SSH is also end to end encrypted)
    - This means that if anyone gets hold of the data as it is enroute from the client to the server, the data will be gibberish
- However, even the endpoint is not guaranteed to be safe (MITM attack)
    - If you are connecting to a server (say google.com) how do you know if that server is actually google.com?
    - Verified SSL certificates are installed on the server
    - For every HTTPS connection, the browser checks the cross-verifies the SSL certificate signature given to it with a Certificate Authority that the browser trusts
    - What CAs does a browser trust? It's a part of the browser itself! You can add your own trusted CAs if you want to
- Summary: Always use HTTPS wherever possible!

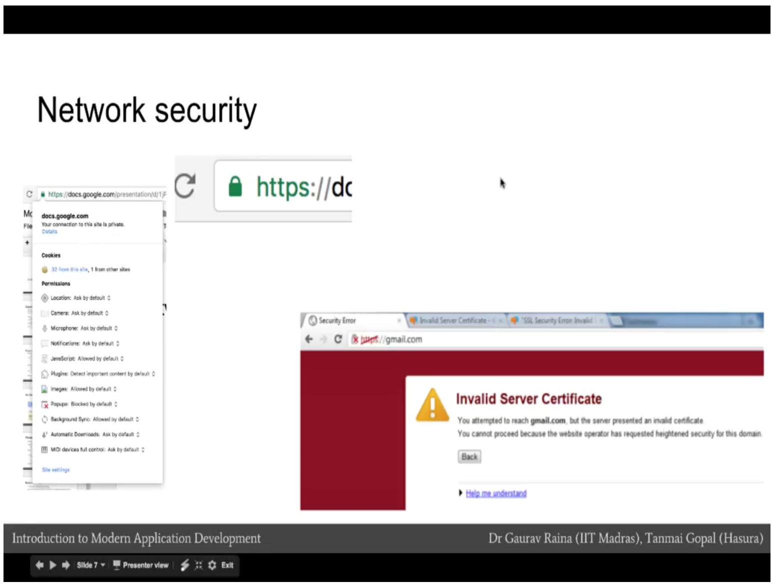
Introduction to Modern Application Development Dr Gaurav Raina (IIT Madras), Tanmai Gopal (Hasura)

Let us come to network security; the first thing about network security is to understand the network is not really owned by us. So, if where the developers we do not own anything on the network. So, anything really can happen on the network and we must be ready for that to happen the most common example of network security breaches are called man in the middle attack click on that link and find out how many in the middle attacks work in how man in the middle attack work and what kind of security breaches they can cause. The only way to protect ourselves despite the fact the network is not on bias is to ensure there is end to end encryption between the client and the server. HTTPS is secured HTTP which means that HTTPS basically HTTP, but with end to end encryption so that the network in the middle if compromise and or somebody is trying to read a traffic in the middle they cannot read the data because the data is encrypted.

The browser creates and HTTPS connection and is able to encrypt data to send to the server by using SSL certificate set have been installed in the server. However, the interesting thing about end to end encryption is that while we can somehow ensure that the data that send from a client to a server is encrypted, how can we ensure the service actually the server. For example, when we connect to Google dot com, how do we even know whether the server is actually Google dot com and this is where the second more important property of SSL certificate come in where SSL certificate are actually used to identify that a domain belongs to the server that is responding to our request and the way this works is that whenever browser makes an HTTPS connection the browser cross verifies the SSL certificate that is sent to it by the server with a certificate authority, and the certificate authority is are inbuilt into the browser and we can actually update the certificate authorities are add or one certificate authorities that we want, but the browser uses these certificate authorities to check if the SSL certificate given by a server actually belongs to the domain that we making a request to.

So, to summarize the easiest and in a sense the only way to (Refer Time: 10:16) against man in the middle kind of breaches is to use HTTPS wherever possible and to ensure end to end encrypted connection between our application on the client and our application on the server.

(Refer Slide Time: 10:28)



Network security

https://docs.google.com/presentation/d/1

docs.google.com
Your connection to this site is private.
Details

Cookies
32 from this site, 1 from other sites

Permissions
Location: Ask by default
Camera: Ask by default
Microphone: Ask by default
Notifications: Ask by default
JavaScript: Allowed by default
Plugins: Detect important content by default
Images: Allowed by default
Popups: Blocked by default
Background Sync: Allowed by default
Automatic Downloads: Ask by default
MIDI devices full control: Ask by default

Site settings

https://dc

Security Error | Invalid Server Certificate | "SSL Security Error Invalid

~~https~~://gmail.com

Invalid Server Certificate

You attempted to reach **gmail.com**, but the server presented an invalid certificate.
You cannot proceed because the website operator has requested heightened security for this domain.

Back

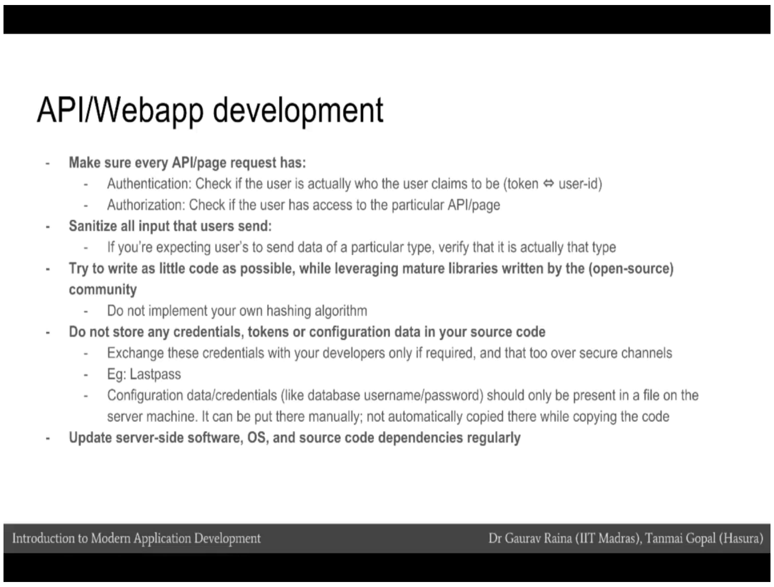
Help me understand

Introduction to Modern Application Development | Dr Gaurav Raina (IIT Madras), Tanmai Gopal (Hasura)

So, this is why browsers have started displaying a little green icon next to the HTTPS symbols. So, whenever you go to a site that has HTTPS in able to you see a little green icon, and if you click on that green icon I drop down and it will tell you what the details about this SSL certificate, and what the difference sub domains that are covered under the SSL certificate.

On the other hand if you have any HTTPS connection to the website, but the SSL certificate has not been verified or the verification has failed with the certificate authority, then you will get an error like this. So, for example, in this case the server might not actually be Gmail dot com and some other server in the middle on a network is pretending to be Gmail dot com, and we have an HTTPS connection with that server with that server is not Gmail dot com.

(Refer Slide Time: 11:12)



API/Webapp development

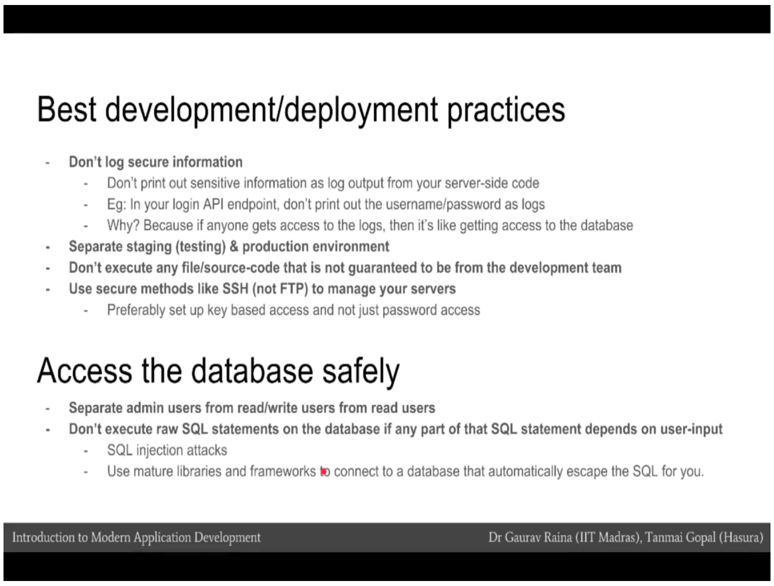
- **Make sure every API/page request has:**
  - Authentication: Check if the user is actually who the user claims to be (token ⇔ user-id)
  - Authorization: Check if the user has access to the particular API/page
- **Sanitize all input that users send:**
  - If you're expecting user's to send data of a particular type, verify that it is actually that type
- **Try to write as little code as possible, while leveraging mature libraries written by the (open-source) community**
  - Do not implement your own hashing algorithm
- **Do not store any credentials, tokens or configuration data in your source code**
  - Exchange these credentials with your developers only if required, and that too over secure channels
  - Eg: Lastpass
  - Configuration data/credentials (like database username/password) should only be present in a file on the server machine. It can be put there manually; not automatically copied there while copying the code
- **Update server-side software, OS, and source code dependencies regularly**

Introduction to Modern Application Development   Dr Gaurav Raina (IIT Madras), Tanmai Gopal (Hasura)

Now, let us move on to some best practices for API and web app development, whenever we write an end point that is an API or a web page request then we must ensure that the API has an authentication check and as an authorization check. So, we must check that if a user is claiming to be a particular id then the user actually has that id and we must check that whatever functionality is being requested for in the API or on the page is something that that particular user id actually has the authorization to make a request to. We should also always sanitize all the inputs that users send and we should never trust the input that is sent by user the structured the format or the size of the data that is being sent by the user.

We should also try to write as little code as possible to achieve the functionality that we want, we should use as many mature library that have been written by pressure built the open source community as possible. For example, we should never implement our own hashing algorithm which might have bugs and which might not be as secure as existing libraries. We should not store any credentials tokens or configuration data in a source code for example, in a web app course code we should not write the database username and password that a web app is using to connect to the database server. We should create the files that contains this configuration data directly on the server, and not store them in a git repository if you need to share this is credentials amongst other developers in an organization, we should use a service like lastpass to share this credentials. Once again just like with front end code we should regularly update server side software the operating system and other libraries that are source code depends on.

(Refer Slide Time: 12:48)



Best development/deployment practices

- **Don't log secure information**
  - Don't print out sensitive information as log output from your server-side code
  - Eg: In your login API endpoint, don't print out the username/password as logs
  - Why? Because if anyone gets access to the logs, then it's like getting access to the database
- **Separate staging (testing) & production environment**
- **Don't execute any file/source-code that is not guaranteed to be from the development team**
- **Use secure methods like SSH (not FTP) to manage your servers**
  - Preferably set up key based access and not just password access

Access the database safely

- **Separate admin users from read/write users from read users**
- **Don't execute raw SQL statements on the database if any part of that SQL statement depends on user-input**
  - SQL injection attacks
  - Use mature libraries and frameworks to connect to a database that automatically escape the SQL for you.

Introduction to Modern Application Development Dr Gaurav Raina (IIT Madras), Tanmai Gopal (Hasura)

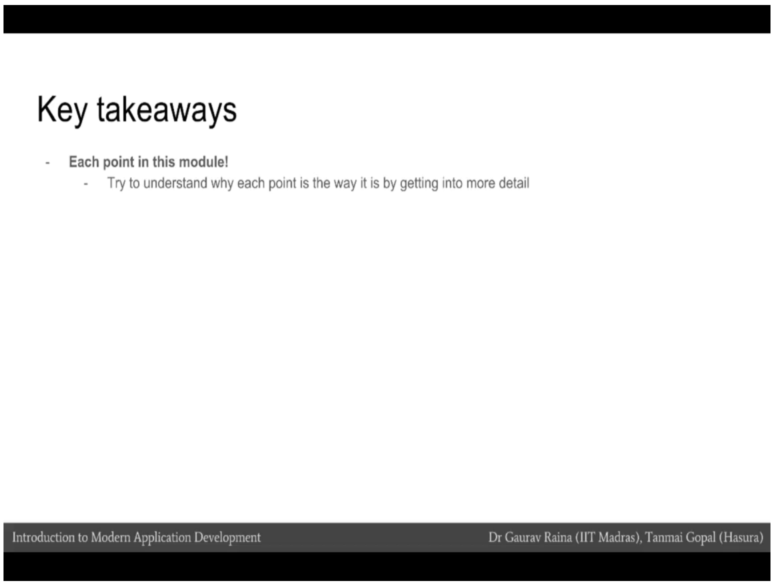
When we are deploying our source code then there are few things that we need to make sure that we take care of for example, our server might be outputting some logs we might be printing out some logs as a program executes primarily for debugging purpose, we should remember never to print out logs that contain sensitive information.

For example in a login end point we should not log the username and password that is being sent by the user, because the whole point it defeats the whole point of hashing which was to store the sensitive data in a secure way, but in this case logs will be stored that contain the wrong username and password. So, if anybody ever get access these logs then that entire data is compromised. We should also try to separate the staging and production environment as much as possible so that we can test thoroughly and during testing we can have a mode last attitude toward security, but during production we can take greater care of security so that debugging is easier on the testing environment, but the production environment is secure. When we are deploying source code or we are deploying binary is that our server side code on our machines we should ensure that we are not deploying or executing any files that are not guaranteed to be from our own development team or a not guaranteed to be community libraries or operating system dependencies.

We should use secure methods like SSH or to manage your servers for example, FTP is a popular method to upload files to server this is not secure. SFTP is a more secure version of FTP, even when we use SSH there is a method of setting up key based access and not just using password to access because if use password to SSH remote servers then we have to worry about sharing the password. When we need to access a database we should ideally separate the administrative use of the database from users that have read write permissions. So, for example, the webapp can use the read write username and password for the database. If the webapp is just reading data from the database then we should create a separate read user and give the webapp the read username and password to access the database.

So, this way we can ensure that even if the webapp is compromise completely they would not have more access that are strictly necessarily when they talk to the database. We should also never execute raw SQL statements on the database if any part of that SQL statement depends on user input, and this leads to kind of attacks called as SQL injection attacks which are the most common way of attacking databases.

(Refer Slide Time: 15:17)



Key takeaways

- Each point in this module!
  - Try to understand why each point is the way it is by getting into more detail

Introduction to Modern Application Development Dr Gaurav Raina (IIT Madras), Tanmai Gopal (Hasura)

And obtaining access to the database that allows hackers to download or delete contact from database. The key take away from this module are unfortunately each and every point in this module, it would be include if you spent time understanding by each point is a valid security point and you can get into more detail about how they actually work; there are several useful resources on the internet that you should browse to understand this topic in more detail.