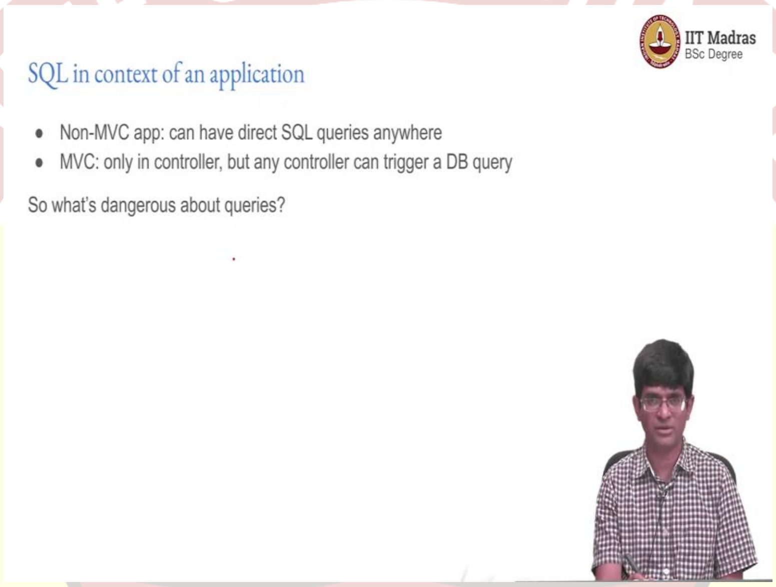
IIT Madras

ONLINE DEGREE

Modern Application Development 1
Professor. Nitin Chandrachoodan
Department of Electrical Engineering
Indian Institute of Technology Madras
Security of Databases

Hello, everyone, and welcome to this course on modern application development. Now, the last part about this session on databases, relates to so called security and this is also relevant to the construction of an app itself.

(Refer Slide Time: 00:30)



SQL in context of an application

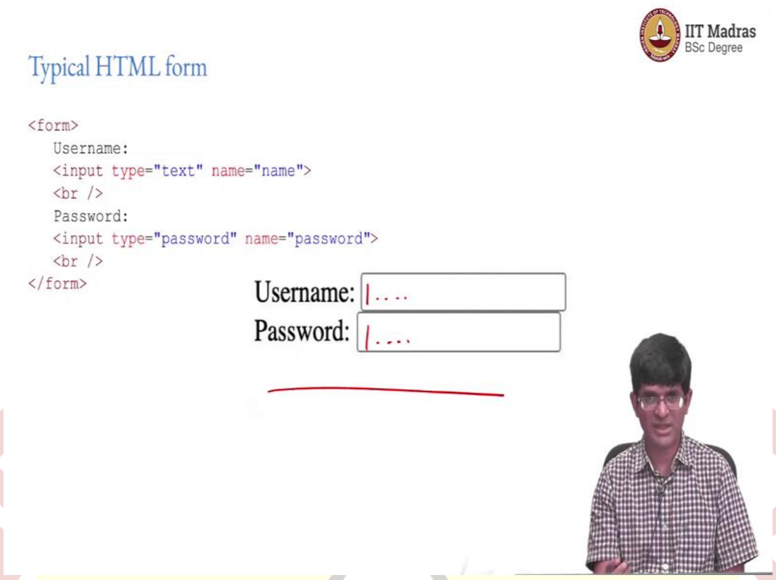
- Non-MVC app: can have direct SQL queries anywhere
- MVC: only in controller, but any controller can trigger a DB query

So what's dangerous about queries?

Now, in the context of an application, you ultimately are querying something from a database and if this was a so called non MVC app, because after all, remember MVC is only guideline. It is not enforced, it is not that you can only build web applications with the MVC framework, you could just as well have a PHP script, which directly queries from a database and draw something on the screen. You could have SQL queries anywhere.

In MVC, of course, you know, ideally, you want it the SQL where it comes, not directly even in the controller, it is only in the model. So, the controller sends the queries, constructs the queries and sends them to the model and the model is the part that actually handles communicating with the database. So, if at all there is SQL involved, it is going to be only in the model part, not really even the controller section. So, the question becomes what is dangerous about an SQL query?

(Refer Slide Time: 01:31)


Typical HTML form

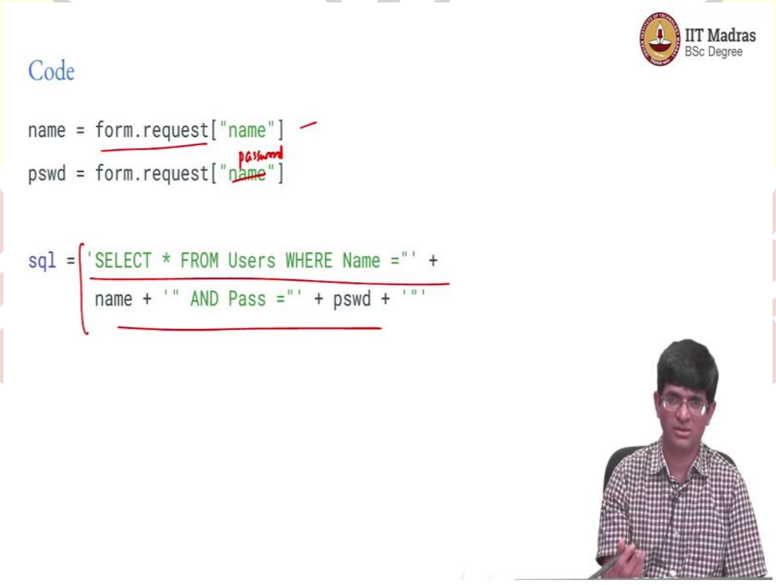
```
<form>
   Username:
   <input type="text" name="name">
   <br />
   Password:
   <input type="password" name="password">
   <br />
</form>
```

Username: |....

Password: |....

So, let us take an example. This is what a typical HTML form could look like. Basically, you have input, for name and input for password, it would get rendered something like, what is shown out here. Which means that finally, you would go and you would type some information for the username, you would type some information for the password and either hit enter or there will be a submit button that you click on.

(Refer Slide Time: 02:01)


Code

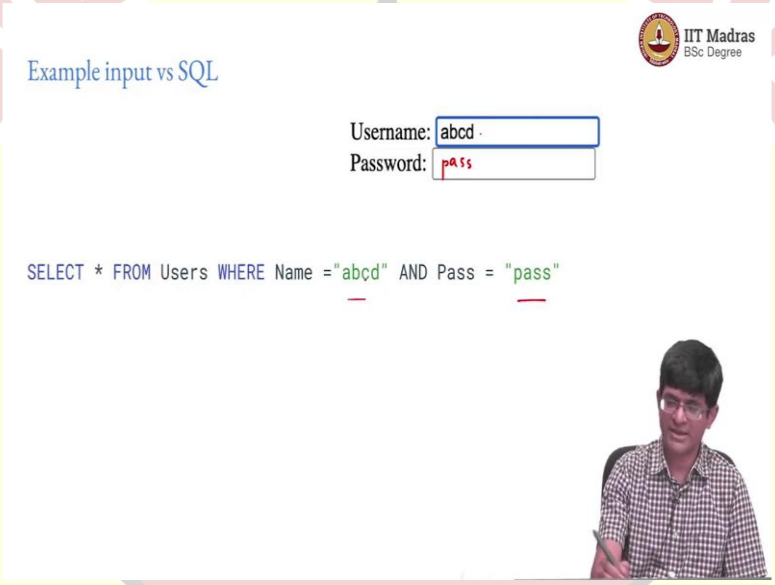
```
name = form.request["name"]
pswd = form.request["~~name~~ password"]

sql = 'SELECT * FROM Users WHERE Name ="' +
      name + '" AND Pass ="' + pswd + '"'
```

What happens once that is done is that you will typically have some code inside your script, let us say you are using Python, you might, take form.request[“name”] form..request or either should be password and you are going to construct a query out of it. Now, let us say that your query was just being done by, stringing things together, use the Python + operator and just, concatenate strings. Seems like a reasonable thing to do, potentially very bad. Why? Because you have just directly taken whatever was sent in through the form and you have put it into an SQL query.

Now, who has access to the form, pretty much anyone who has access to that website, not just that, you even cannot be sure that someone actually went to the form and typed it in, they might even just construct an HTTP query directly using curl or something like that and send it to your server. The server has no way of knowing what the client was actually doing. That is the whole point of HTTP stateless condition.

(Refer Slide Time: 03:07)



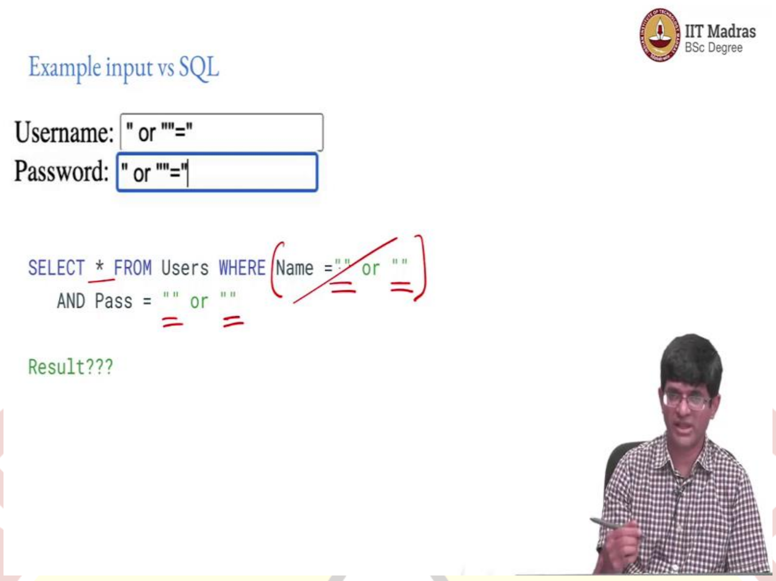
Example input vs SQL

Username: abcd
Password: pass

```
SELECT * FROM Users WHERE Name ="abcd" AND Pass = "pass"
```

So now, what happens if I just type in abcd and you know, let us say I had put in pass over here for the password. It would basically do select * from users where name=”abcd” pass =”pass”, just interpolate both of those perfect, nice save query.

(Refer Slide Time: 03:24)



Example input vs SQL

Username: " or ""="

Password: " or ""="

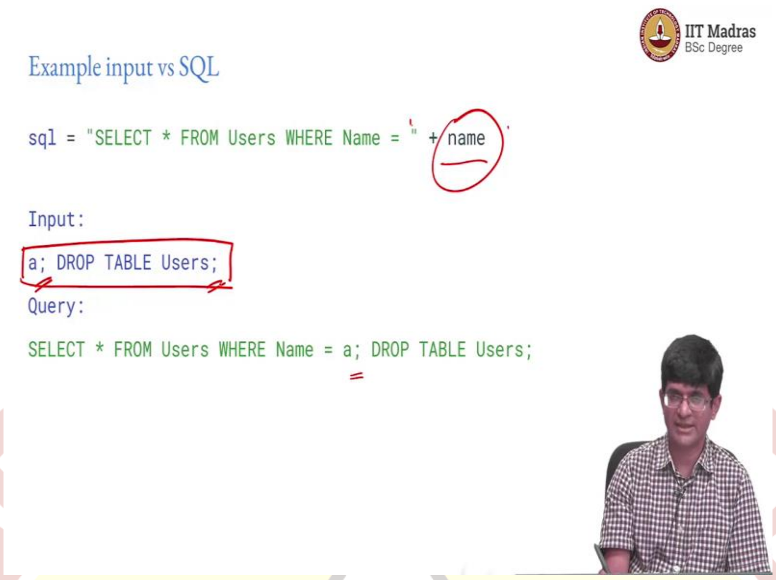
```
SELECT * FROM Users WHERE Name ="" or ""
   AND Pass = "" or ""
```

```
Result???
```

But what happens if I actually typed in some stuff like this, double quotes or double quote, double quote equal to double quote, it looks messy, why am I even typing something like this? But look at what happened, I have now got a query, which says select star from users where name equal to blank or blank and pass equal to blank or blank.

What does this name=“” or “” mean? It basically effectively resolves into a null condition, because it is not even going to look for the name and what will end up doing is just selecting * from users and so, blank or blank will basically just, this part of it will just, turn out into an true statement and that is it. So, you would end up getting all the entries from the database. So, the result is that you have leaked information, basically something which was supposed to be in the database, which someone else was not supposed to be able to see has come out as a result of this query.

(Refer Slide Time: 04:34)



Example input vs SQL

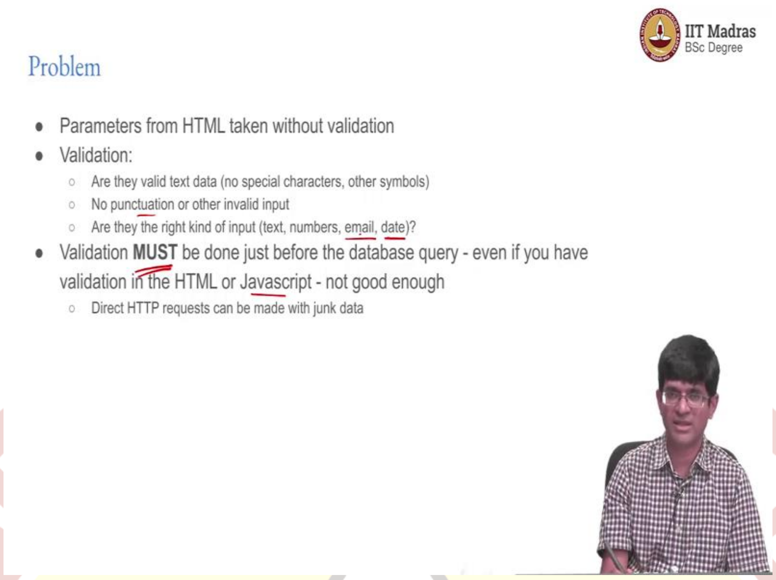
```
sql = "SELECT * FROM Users WHERE Name = " + name

Input:
a; DROP TABLE Users;
Query:
SELECT * FROM Users WHERE Name = a; DROP TABLE Users;
```

Now, even worse, is let us say that you did not even have the sort of single quotes and so on over here, you just straightaway do name equal to this plus name. What happens if I give an input like this within my box over here, I basically say a semi colon, drop table users semi colon. What is the query that is constructed select * from users where name = a; drop table users, they are selected separated by the semicolon, they will be treated as two separate SQL queries and both will run result is total destruction of your data, what is the problem?

The problem was that this name, this parameter came in from outside and you did not validate it, you did not sort of check in any way, whether it had, these extra characters, all these semi colons, they were the root of the problem, That is what allowed somebody to construct two queries like this. And as a result of that, it just basically constructed something which should not have been possible.

(Refer Slide Time: 05:36)



Problem

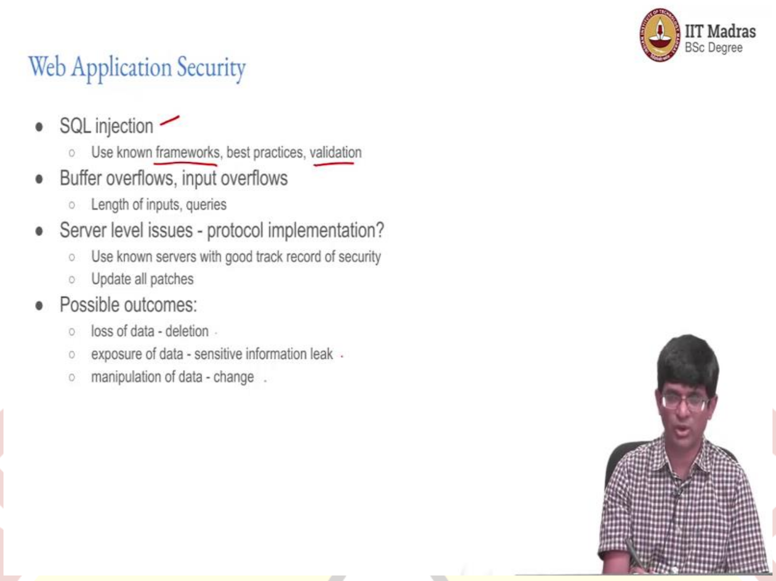
- Parameters from HTML taken without validation
- Validation:
  - Are they valid text data (no special characters, other symbols)
  - No punctuation or other invalid input
  - Are they the right kind of input (text, numbers, email, date)?
- Validation **MUST** be done just before the database query - even if you have validation in the HTML or Javascript - not good enough
  - Direct HTTP requests can be made with junk data

Now, the parameters from the HTML were taken without validation and validation should not go through multiple things, I mean, it should basically check first of all, are they valid text data? No, especially characters or other symbols. In particular, nothing like semi colons or quotation marks, they should not be there or if they are, they have to be sort of escaped out into something which, looks sufficiently different that the SQL query will not get confused.

You do not want any punctuation or other kinds of invalid input and you could also have an extra level of validation, which actually checks I mean let us say that it is supposed to be an email, does it look like an email, does it have one front part, an add symbol @ symbol which looks like a domain name after it?

If it is supposed to be a date? Does it look like a date? So, all of this validation must be done just before the database query, what do I mean by that, you have to be very careful that you do not assume that just because you have, let us say, web page, which is behind some secure firewall or whatever it is and you have told people to enter the correct information, that the data you get will be correct, because somebody might still figure out a way to send a query directly to your server, without going through the form that you created and as we will see later, you might have JavaScript which is being used for validation. But let us say that the person has directly constructed the query, they are just going to bypass the JavaScript altogether and create a query that comes into your server and destroys data on it and you have no way of stopping.

(Refer Slide Time: 07:18)



Web Application Security

- SQL injection
    - Use known frameworks, best practices, validation
- Buffer overflows, input overflows
    - Length of inputs, queries
- Server level issues - protocol implementation?
    - Use known servers with good track record of security
    - Update all patches
- Possible outcomes:
    - loss of data - deletion
    - exposure of data - sensitive information leak
    - manipulation of data - change

So, web application security, in other words needs and can have a lot of different variants. In fact, this is probably the subject of a course in itself, what I described over here is what is called SQL injection, where you are basically trying to inject some invalid or bad constructs into the SQL query. The best way to get around it is to use known frameworks and good solid validation that has been provided, usually that has been tested in the field, you might write your own validation, but there is always the chance that you have missed something out, so it is better.

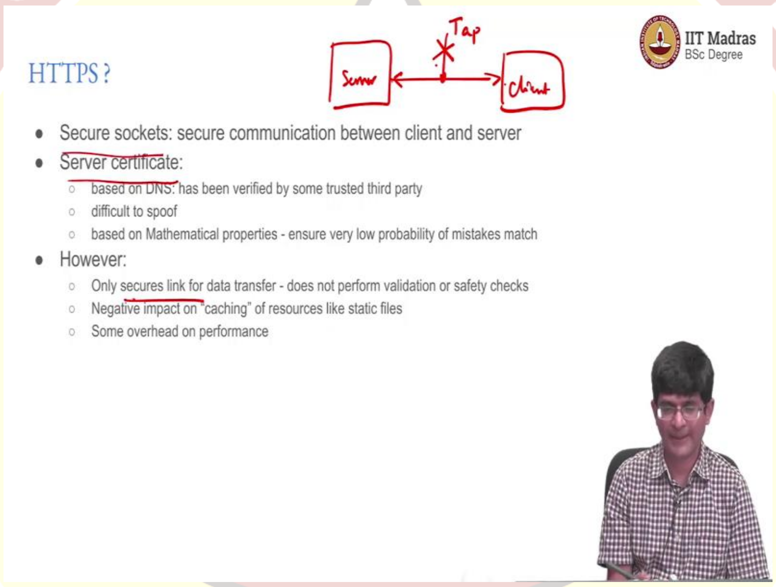
That is one of the chief reasons for using known frameworks, even though you feel that you might know what kind of problems are there, there might be something else that gets discovered, it will get fixed in the frameworks first, because a lot of people are using them and you might miss it and not really be able to accommodate it in your app.

There are other things called buffer overflows and input overflows, which are basically related to the length of the query the URL and so on, which can sometimes even crash different clients. So, specific types of clients might crash because of this or specific types of servers might crash? What happens to the protocol implementations of servers? I mean, are they able to accept any kind of query, what happens if I inject some unknown characters into the middle of a query, into the middle of a request, which is being sent to a HTTP server.

And one thing, actually that are not mentioned here. Remember the character set and all the encodings that are used, the reason why we have character sets is of course, so that we can use different languages like let us say, Hindi or Tamil. But the problem is, I can also have different characters or different entries from different character sets that look very similar to things that I am already familiar with, but which can actually end up crashing a server.

So, all that validation has to be done right at the last step before it hits the database. So, that you are sure that only valid data is going to get inside the database. Possible outcomes can be serious, I mean, you can lose data, you can expose sensitive data or you can even change data without knowing it and all of these are potentially deadly to whoever is running the app. Because let us say that you are, even leaking, let us say credit card information or changing the date of birth of a person, all of those can have very drastic effects on other things that you may not even think about at the time.

(Refer Slide Time: 10:01)



HTTPS ?

- Secure sockets: secure communication between client and server
- Server certificate:
  - based on DNS: has been verified by some trusted third party
  - difficult to spoof
  - based on Mathematical properties - ensure very low probability of mistakes match
- However:
  - Only secures link for data transfer - does not perform validation or safety checks
  - Negative impact on "caching" of resources like static files
  - Some overhead on performance

Now, a word about HTTPS literally, this is I am not really spending too much time on it. Because at this point in time, pretty much HTTPS is something which you should think of as automatically, pretty much, anything any app should use or any web server should pretty much use HTTPs and the reason for that is, all that it does is let us say that you have a server and you have a client, there is this pipe of data which is flowing between them, the server can, the client can send request to the server, the server sends back responses to the client, you can think of it as a pipe with data flowing across it.

Now, in pure HTTP, a third party can tap into this somewhere, it could literally be putting a tap on the wire or it might be that, you know, it is passing through some intermediate router and I am able to look at it in one of the routers, the packets are there and I just basically reassemble the packets at one end and see the data going back and forth.

Which means that a person who is taping over here can see all the information going between server and client. What HTTPS does is it provides something called a secure socket layer, which essentially says that this tap becomes impossible. Mathematically impossible at least, mathematically very, very hard to do. That is probably the way to put it.

And, effectively, what it is saying is that, once HTTPS with the kind of protocols, kind of servers, kind of clients is in place, it is now extremely difficult for a third party to tap into this and be able to know, what is the communication happening between the client and the server. There can be instances where you do not care about it. But by default, pretty much it is usually safer to assume that HTTPS is the better way to go.

Because the moment you have anything down to something like a password, it has to be kept safe and, in fact, nowadays, you will notice that you even browsers have extensions or variants called HTTPS Everywhere, which tries to switch you over to an HTTPS version of any website. Because the assumption is that by default, you should not be sort of using plain HTTP.

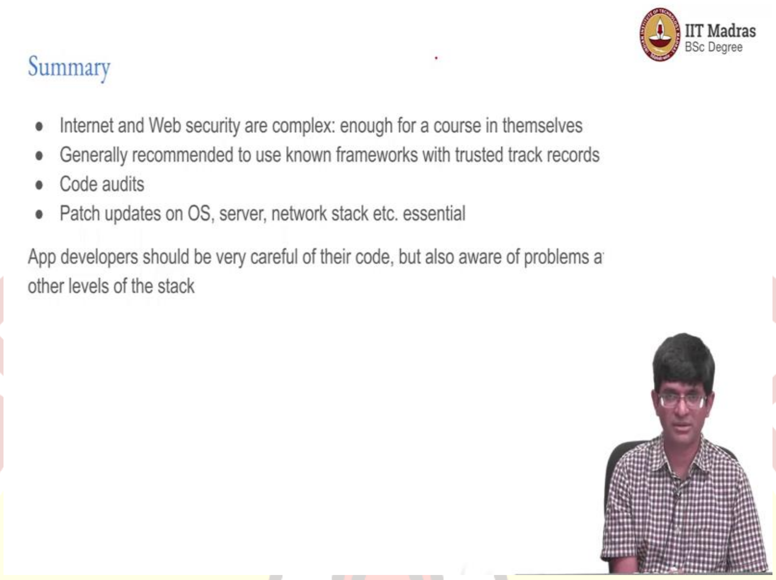
Even by accident, you do not want to be leaking information. How does HTTPS work? There is a lot of theory behind it and the important thing is there is something called a server certificate, which is, what results in that small green sign on the URL bar and that server certificate has been essentially verified by some third party who is trusted both by the server and by you, by you meaning whoever created your operating system.

Very difficult to spoof based on mathematical properties, which ensure very, very low probabilities of accident, accidental mismatches or something. But the important thing to remember over here is that it only secures the link for a data transfer, it does nothing about the data which is going through.

So, in particular, it does not perform validation, safety checks, nothing of that sort. All that you can say is nobody can tap into your information, they can still feed you junk. One problem with HTTPS is that, previously when plaintext data was being sent back and forth, intermediate proxies could sit over there and say, this is what you are asking for here. I have it. Let me give you back the data.

Now, the intermediate proxy, by definition cannot see what is inside the request that you are sending. So, it cannot send you back information, even if it has it. So, it has a pretty big negative impact on the cache ability of resources like static files and also some overhead on performance itself.

(Refer Slide Time: 14:00)



Summary

- Internet and Web security are complex: enough for a course in themselves
- Generally recommended to use known frameworks with trusted track records
- Code audits
- Patch updates on OS, server, network stack etc. essential

App developers should be very careful of their code, but also aware of problems a
other levels of the stack

So, to summarise this part on security, internet and web security are complex beasts, they are pretty much enough for a course in themselves and right now, as an application developer, the main thing that you should be thinking about it as, as far as possible, use known frameworks with clustered track records.

But also, be aware of where your application is going to run, are you running it on your own server or are you running it on something like Google App Engine, if it is something like App Engine, or slightly better off because they are taking care of all the problems of running the server, maintaining the security patches on the server, preventing things like buffer overflows and things of that sort.

So, you have to concentrate only on your own validation so that your database does not get messed up. So, as an app developer, in other words, you should be aware of all possible problems that arise from the code that you have written. But also need to be aware of problems at other levels of the stack and by stack, what I mean is, your application is finally running on top of, some kind of a server or language interpreter, which is running on an operating system which is running on some kind of hardware which is running in a data centre.

So, there are many, many different stages before your clients or users actually hit the application that you have written. So, at this point, the main thing to keep in mind is there are such things as SQL injection and various other kinds of attacks that you need to be aware of, and make sure that cannot happen in the code that you write. But the rest of it is also something that you need to keep an eye on. You need to know where you are running your application, so that it is actually going to be safe.