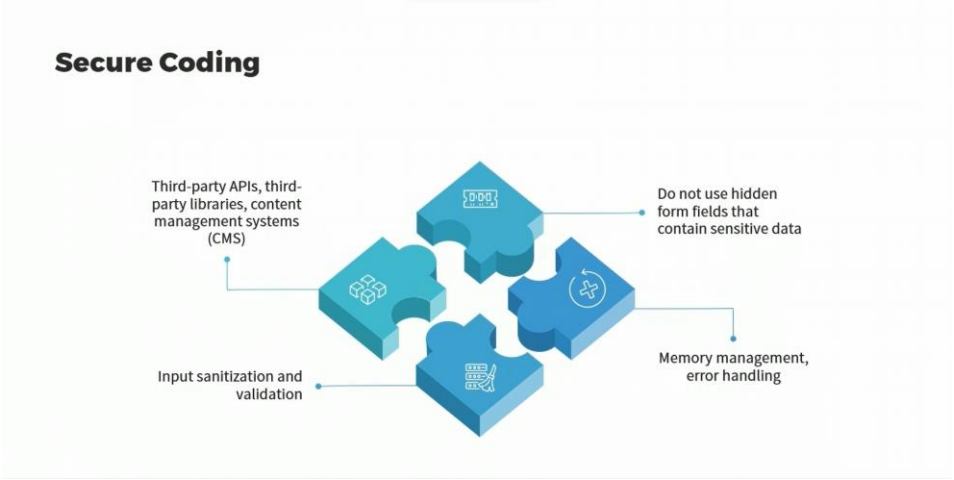
This PDF Lecture is made by
Abid Biswas, Ethical Hacker **(Purple Teaming)**
(CEH, NSE1, NSE2, NSE3,
$ISC^2(CC), Google\ Cyber\ Security\ Professional\ Cert$)
BSc in Information and Communications Engineering (ICE) at EWU

One way to ensure you have a secure web application is to make sure that web application developers are adhering to secure coding guidelines. This means that we can reduce the likelihood of vulnerabilities, whether it's in firmware code, or in our case, for a web app, software code. It means training for software developers in how to code securely, such as which libraries that are trusted should be used and how, perhaps to call functions and how to deal with return results, and so on. Secure coding is important also in the sense that it needs to be included in all design and development phases.

It's not something that should be thought of after a web app is developed and we decide that we need to secure it. It should be something that is thought of right from the design phase at the very beginning. It means using only trusted components and secure libraries for your web application. Web app developers seldom will create everything completely and entirely from scratch. They might use existing code libraries or components that perform certain tasks needed by the web app instead of recreating the wheel.


It speeds up software development, but it needs to be trusted to make sure that those items don't contain any kind of malicious code or will affect the stability of the web application. So, this means then that secure coding for sure involves more effort. It takes longer than if you just didn't care and wrote code freely. Now it might take more time to develop a solution, but in the end, it takes less time than if you didn't do it and had to address security issues over and over and over after the fact. So, with secure coding, we are talking about software developers that are using third-party APIs.

An API is an application programming interface. It allows software developers to hook into other code elsewhere that gives them functionality that they need. Maybe to check stock quotes over the Internet or maybe to connect to a storage account in the Microsoft Azure cloud. There are a third-party code libraries that you can call upon. Again, specific code libraries that might provide the developer with very specific functionality that otherwise wouldn't be available. Software developers might also use content management systems or CMS.




That might even be to the degree where it allows end users to create content for a web app in a friendly interface without exposing the underlying intricacies that would be handled by the CMS environment. Another aspect of secure coding is to make sure that you don't store sensitive information in a way that it can be easily retrieved. Such as using hidden form fields setting a type of hidden is not a great way to protect sensitive information. Yes, visually it's not shown to the user, but there are easy ways and freely available tools with that could be acquired.

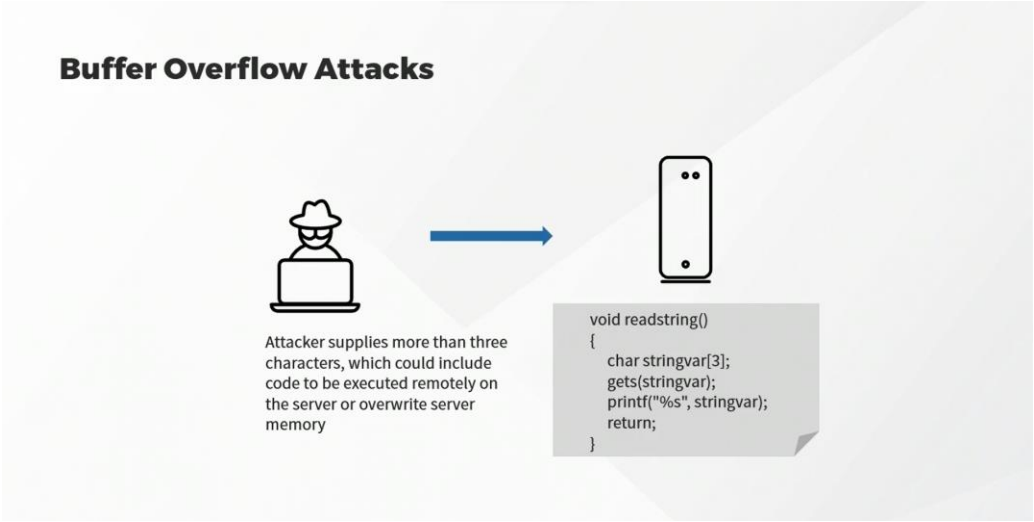
If you don't want sensitive data to be visible then don't include it in an HTML field, maybe encrypt it, put it in a cookie, whatever the case might be. The other thing to think about is to ensure that you have proper memory management. Only allocate the absolute necessary required amount of memory to accommodate something that will be stored in memory, especially when it comes to accepting user input and proper error handling to make sure that despite the best software testing, no software is perfect, there will always be at some point a runtime error.

We want to make sure that error messages don't disclose too much information. We want to try to shut down an app as gracefully as possible, and if we've got an error on a database query, for example, we don't want to display the version of the backend database being used. And then there's input validation and sanitization. These are not the same things, but it's very important to treat all user supplied data as untrusted. Validating input from a user, maybe it's something a user enters into a field on a web form, means checking that it is valid.

For example, if we are accepting a Canadian postal code or an American ZIP code into a field, we should validate it to make sure that the correct number of characters are entered and the correct types of characters were entered in the right positions. So, all we're doing with validation is checking, maybe even checking that all mandatory fields at least have been filled in. But input sanitization actually modifies user input. Usually, we use input sanitization to remove special characters, maybe to remove double quotes or replace them with single quotes, or maybe remove special characters such as less than and greater than signs from input fields to help mitigate against things like stored cross site scripting attacks. All of that is very important.

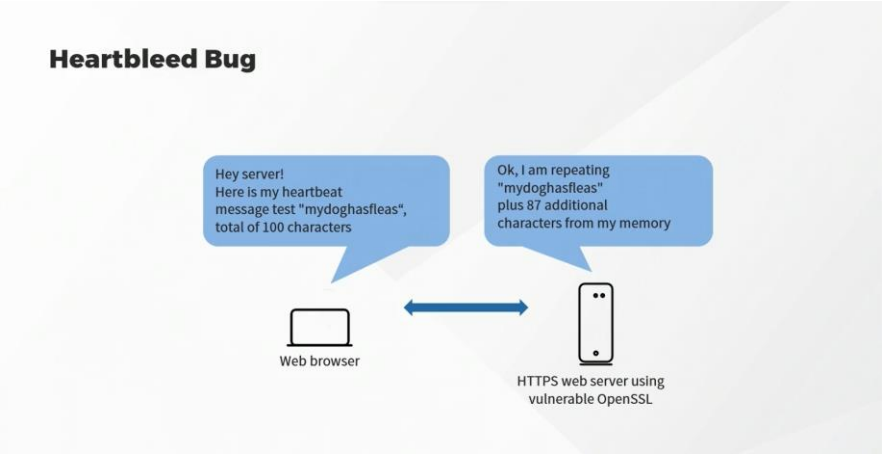
When it comes to memory management, buffer overflow attacks come to mind. These have been around for decades, and they still persist, although in different forms. Let's say that we've got an input field on

form that expects the user to enter 3 characters, maybe it's a cost center code. But an attacker supplies more than 3 characters because we haven't enforced that in our coding.



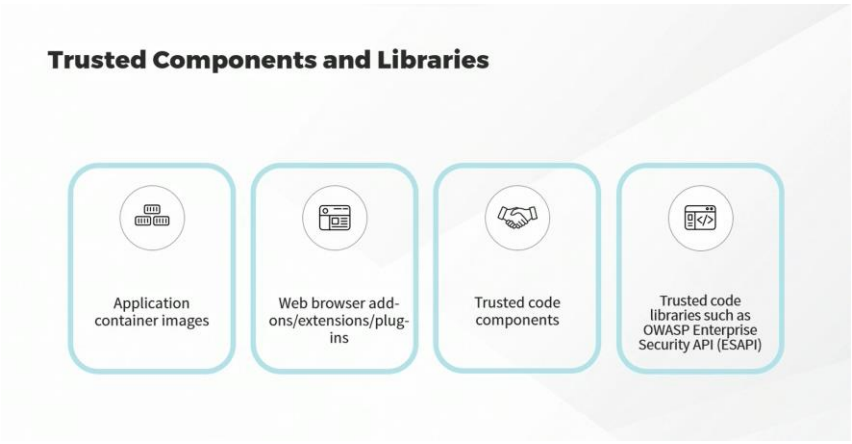

So, what the attacker supplies beyond those 3 characters might include code that's designed to be executed remotely on the web server, or maybe it will overwrite server memory which could cause a problem like disclosing sensitive information in server memory or affecting the stability of the server. On the right in our diagram, we have some code that would be running server-side where we've got a variable called stringvar. The type is character or char, so this is like C code and we are setting it up as a 3 character variable. Then we're using the gets function to retrieve user input into that string variable and memory, and then we're using printf to display it back.What's not happening here is we aren't verifying the length of what was supplied by the user or anything like that, so this is an example of poor coding.

Another variation of this that was quite popular years ago was the Heartbleed bug. The Heartbleed bug affected older versions of open SSL, which was ironically used to secure communications to and from web servers. So, in our example on the left, we've got the attacker's web browser that is sending what is called a heartbeat message to the server. Heartbeats are common in a lot of different technologies.



They're also called **keep alive packets to make sure that other devices over the network or clients are still there and that the session should remain open**. So, in this example, the malicious web browser will send a heartbeat message, let's say, of the text mydoghasfleas, which is 13 characters long. But when this is sent to the server, we're telling the server, it's 100 characters long, it's not, it's only 13. If that's not being checked server-side, then the server will say, OK. I'm going to repeat back your data in the heartbeat response.

I'm going to repeat back mydoghasfleas, which is 13 characters long, but you told me it was 100, so I'm going to give you 87 additional characters from memory, which could disclose sensitive information, especially if you run this over and over and over and over. You can essentially build a map of what might be in the server's memory. Now, this type of problem, of course, is fixed normally by applying a patch for the vulnerable component, in this case, the OpenSSL libraries. Secure coding means using trusted components and libraries and what forms those come in, application container images.




When you run an application container, you run it from a container image and you need to make sure your container images are acquired from trusted sources. Web browsers with add-ons or extensions or plugins to give them extended functionality if that's what's being used for a web application. So, you have some server side components and you also have a web browser add-on component. If you're using something that's already been built by another party. You want to make sure that you test it thoroughly.

Make sure you know the documentation and whether there are patches for that component, make sure it's all up-to-date. So, there might be other trusted code components like libraries that your code is calling upon to run functions. Again, needs to be from a trusted source, and you can use trusted code libraries like the OWASP Enterprise Security API, otherwise called ESAPI, to make function calls to do things like validate and sanitize input. So, yes, secure coding is a big deal, but it doesn't solve all of your security problems with web apps. It doesn't address things like a lack of security being adhered to during all software development phases.

It doesn't deal with poor OS security or HTTP stack security configuration flaws or app configuration flaws. Secure coding isn't a mitigation against Denial of Service or DoS attacks, which are used to flood a network or a server with useless traffic, thus preventing legitimate traffic. Secure coding won't help that, a web application firewall could. Distributed Denial of Service or DDoS attacks simply means that we have a multitude of hosts issuing the attack against the target. Now that's not to say secure coding isn't important,

it is. It's just one component in an overall larger security model to develop and maintain secure web applications.