# Html5-quiz

* *There are FOUR (4) questions in this quiz. You need to answer all of them. The questions are based on the paper we discussed in our lecture, "*Code Injection Attacks on HTML5-based Mobile Apps: Characterization, Detection and Mitigation*".*
* *This quiz is open book. You can also use the Internet for searching. You cannot discuss with others about the answers. You must describe the answers in your own words.*
* *There is no time limit for the quiz, but you have only 1 attempt to answer each question.*
* *Limit your answer to two short sentences (20 words or less for each sentence on average) for the questions.*

1. List at least two differences between code injection attacks in web applications and code injection attacks in HTML5-based mobile applications.

Firstly, HTML5-based apps have a much broader attack surface, i.e scanning 2D barcodes, SMS messages and NFC reader.

Secondly, attacks on HTML5-based app can cause greater damage as they have access to system resources/permissions.

1. HTML5-based mobile apps have a much broader attack surface unlike web applications. HTML5-based apps can interact with many forms of entities like other apps and services such as 2D barcode, NFC, SMS and even MP3 players.
2. Attacks on HTML5-based apps can cause much greater damage than web applications as injected code can freely access local resources. This can directly attack system resources and even propagate to other apps and devices. This is unlike a web browser which is sandboxed, preventing injected code from accessing local resources.

2. Do HTML5-based mobile applications need to enforce the Same-Origin policy? Justify your answer.

No. Request can be made from other sources such as the file system, camera, or any other source.

No, the Same-Origin policy would not have much impact in securing HTML5-based mobile applications because code injection channels extend beyond the web origin. The broad attack surface for code injection can be unique to mobile devices like SMS and can even come from internal channels such as the local file system

3. This paper describes a solution using static taint analysis, which tracks how program states flow through an HTML5-based app. Using Figure 4 as an example, describe how this technique can help to identify the vulnerability. In addition, identify the source of taint and the sink of taint in the code in Figure 4, which is shown below.

Text, letter

Description automatically generated

This technique can help to trace the data flow of a source to where the data is being processed. For this example, the source can be identified on line 3 where it gets an input from a barcode scanner, and as for the sink, it is on line 6 where it is printed out.

Static taint analysis can help identify a vulnerability by first focusing on the sink, which are call sites of unsafe API. Then working backwards from the sink to find out how its input is formed is called the taint. This analysis can help identify what the code injection channel is and how the malicious code will be triggered. In this example, line 3 is the taint as the malicious input can come from a QR code with embedded malicious code. Line 6 is the sink which will run the malicious code from the QR code input.

4. This paper discussed using CSP as a system-level solution to defeat the reported attack in Section 5.3. If the Android system decides to adopt CSP as the solution, where will the CSP policy be specified (note that there is no HTTP header in hybrid apps)? What changes will be made to the hybrid app’s source code?

It can be specified using a meta tag within the HTML file of the app. The source code will have to include this meta tag in all the HTML files of the app or even generate it dynamically at runtime using JavaScript. Another change would be that needs to be done would be to move inline scripts to external files or the meta tag will have to specify the trusted sources of these scripts.

The current CSP found in WebKit, the basis of Webview, allows to add CSP policy to be specified in the html meta tag. The hybrid app's source code can be changed to include this CSP policies in its meta tag, and also inline Javascript code have to be removed and changed to external javascript files which their hashes matching that in the CSP policy. Any use of eval() in the source code would also have to be removed.