Analysis Report for: Adjuntos-Y8U5s2n2Z7w0q26.PDF.html

This code is not C code; it's HTML and JavaScript embedded within HTML. There's no C code to analyze. The provided code creates a web page that simulates a file generation process. Let's analyze the HTML and JavaScript aspects.

Overall Functionality

The HTML code displays a web page with a loading animation and several seemingly random alphanumeric strings. The animation gives the impression that a file ("adjunto seguro") is being generated. The `iframe` attempts to load a base64-encoded HTML file which contains more JavaScript code that logs messages to the console. This second HTML page seems designed to further obfuscate the actual intent of the script. The overall effect is designed to be deceptive and potentially malicious.

JavaScript Code Analysis (from the base64 encoded iframe)

The base64-encoded data within the iframe decodes to HTML and JavaScript code. The JavaScript section declares numerous variables that appear to be random strings and functions that simply log "Alerta" messages to the console with different numbers attached. This is a clear obfuscation technique often employed in malware to make reverse engineering more difficult.

Function Summaries (JavaScript)

The JavaScript code consists primarily of function declarations, most of which have the following structure:

```javascript
var functionName = function() { console.log("Futu... Alerta number"); }; functionName();
```

Each function's purpose is to log a message to the console. The message is a seemingly random string, possibly a decoy, and a number. There are no parameters, and the return value is implicit (undefined). The `console.log` statement suggests a potential for logging information for later retrieval or for testing.

Control Flow (JavaScript)

The control flow of each function is simple: it executes a single `console.log` statement and then terminates. There are no loops or complex conditional statements within these functions themselves. However, the calling of many functions in sequence creates a simple linear execution flow.

Data Structures (JavaScript)

The primary data structures are strings. The numerous variable declarations store strings that appear random but might encode some meaning if decoded.

HTML Structure

The HTML structure is straightforward. It consists of two divs (`CIQK4631745675736`, `MBI15785994552454`), one showing a "generating" message with a progress bar, and the other displaying more random alphanumeric strings that might be used as identifiers for later analysis. An iframe loads the base64-encoded HTML which further contains more functions that log messages to the console. The CSS styles the page to provide a more sophisticated loading screen appearance.

Malware Family Suggestion

Based on the obfuscation techniques (random variable names, base64 encoding, and the console logging), the deceptive progress bar, and the overall lack of clear functionality beyond logging, it strongly suggests this code is part of a **downloader** or **information gathering** malware family. The numerous random strings may represent components of a larger malware attack. This code likely acts as a first stage, establishing a connection, reporting back the status of the infected machine, and then downloading additional, more destructive components. The lack of clear function names and the use of potentially meaningful base64-encoded HTML obfuscates the exact function. Further analysis of the server to which a connection might be made (via the iframe) would be needed to definitively determine its malicious payload or nature. The use of JavaScript, rather than a more conventional compiled language like C, is frequently preferred by malware authors due to the ease of obfuscation and distribution.