**Introduction**

Websites are composed of three different technologies. HTML takes care of the structure (contents) of the website and CSS is about the style. Finally, JavaScript offers the functionality that let the user interact with the website. Because JavaScript programs run on user's machine, extra care should be taken that they never exploit her private data stored locally. The *same-origin policy* was created for that reason and states that Javascript programs must only be able to see data from the server that produced them, partitioning the web by its users.

However, that principle was never used for the hyperlinks, otherwise we would not be able to see visited and unvisited links rendered differently.

**History sniffing**

The server (website) can probe a bunch of links and test if they are in your history, based on the way they are rendered. This is called *history sniffing*.

There three types of history sniffing:

- Direct
- Indirect
- Side-channel

**Direct sniffing**

JavaScript program can examine and manipulate the page using DOM API. A malicious site can guess URLs of pages that its visitor have visited, creating links pointing to those URLs and afterwards checking how these links were rendered.
**Indirect sniffing**

There are two ways somebody can perform indirect sniffing attacks. First, he can make visited and unvisited links have different size. This will make some elements of the website move around. Inferring the final position of these elements, he can tell which links were visited. Moreover, he could make visited and unvisited links load different images. Checking which images are loaded, is another way to see which links were visited.

**Side-channel sniffing**

The side-channel sniffing attacks use system mechanisms that leak information that it should not. For example it could use the different time taken to load some links. These kind of attacks, that use loading time are called *Timing attacks*.

**Defence**

In 2010, Baron of Mozilla developed a defence that blocks all these automated attacks. For the direct attacks, the CS APIs pretend that all the links are unvisited. To defend against the indirect and side-channel attacks, all links have the same size and take the same time to load.

**Interactive attacks**

Interactive attacks are attacks that need users’ interaction with the malicious site. Unfortunately, Baron does not attempt to address these attacks.

Some examples of interactive attacks are:

- Word captcha
- Char captcha
- Chessboard puzzle
- Pattern matching
Word captcha

In these interactive attacks, the attacker correlates each link he wants to probe with a “captcha-ed” word. Visited links have, for example, black color and the unvisited ones have the same color as the background. Then the victim is asked to complete the words he can see, as if he completed an ordinary captcha.

Char captcha

This works the same way as word captcha. However instead of words, user needs to type seen characters. These characters are like 7-segment and created in such way that each character probes three links.

Chessboard puzzle

The victim is asked to click on the powns he see on a chessboard. These powns are related to a link. Thus the attacker can find out which links are visited, knowing which powns are visible to the victim.

Pattern matching

In this attack the target is asked to click on some images that assembled create a pattern.

Experiments

They created a website and asked 307 participants to interact with such interactive attacks.

Results

According to the results, they found out that word captcha is the interactive attack that is able to exploit the most visited links.

Side-channel attacks experiments-webcam attacks

Another way to infer if a website is visited is using the webcam of the target.
How this attacks work:

1. Ask user permission to use her webcam

2. If a link is visited : set_background( color of choice ), let's say green

3. Use the webcam and see what the color of the room is.

4. Is it green ? Then it is visited...

**Side-channel attacks experiments-webcam attacks - Results**

In controlled environment they achieved 100% accuracy. However, in real life (participants) they achieved ~15%. Not so much better than guessing.