ARIMA

100% SOFTWARE DESIGN

04 JUNIO, 2024

DESARROLLO SEGURO WEB


SSID: CR
PWD: OngietorriTknikara22

ARIMA

01. / INTRODUCCIÓN

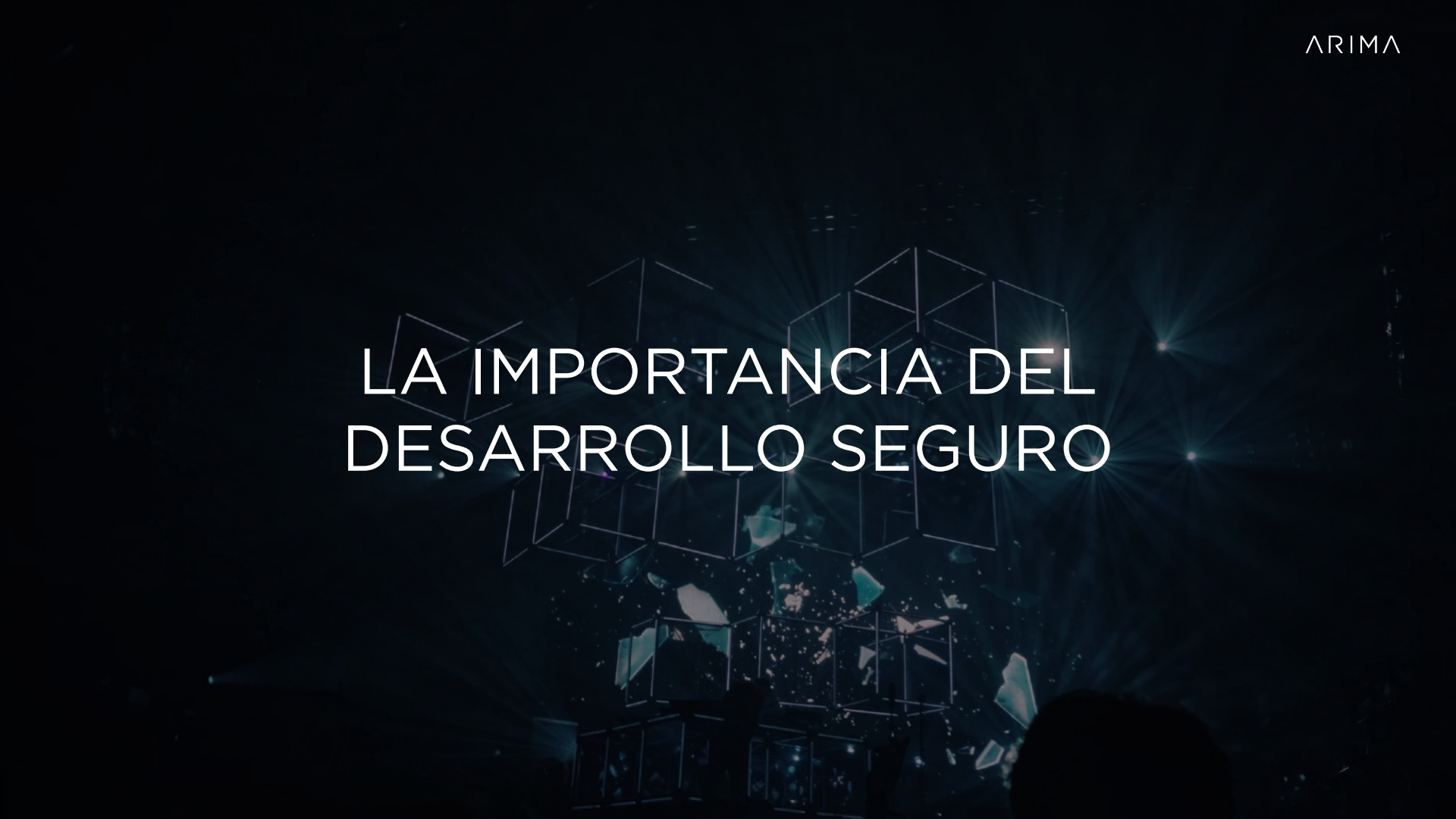
ARITZ
BERASARTE

ABOUT ME

Licenciado en ingeniería informática por la Universidad Pública del País Vasco.
En mis 20 años de trayectoria profesional he tenido la oportunidad de desarrollar y dirigir proyectos full-stack para todo tipo de clientes.
Cofundador de las empresas ARIMA y Hdiv Security.

aritz@arima.eu
https://www.linkedin.com/in/aritzberasarte/
https://twitter.com/Raskasso

¿Y QUÉ HAY DE VOSOTROS?



LA IMPORTANCIA DEL DESARROLLO SEGURO

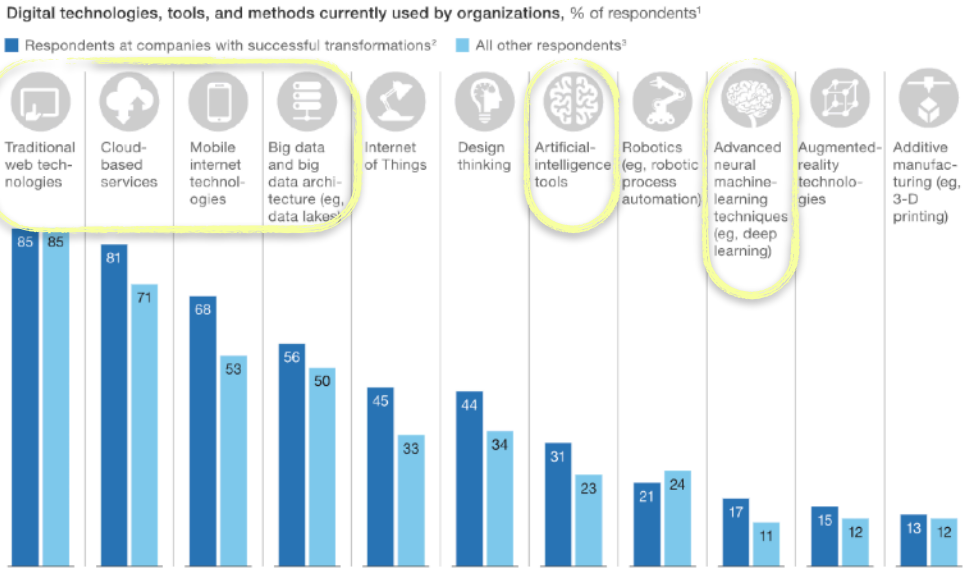
¿DESARROLLO SEGURO?

La transformación digital es un fenómeno global que hoy en día afecta a todos los sectores.

Este proceso de transformación es un gran reto para las empresas pero a la vez también ofrece oportunidades únicas para crecer, expandirse y explorar nuevos caminos.

El software tiene una importancia capital en este proceso de transformación.

LA IMPORTANCIA DEL SOFTWARE EN LA TRANSFORMACIÓN DIGITAL



[1]Respondents who answered "other" or "don't know" are not shown.
[2]Respondents who say their organizations' transformations were very or completely successful at both improving performance and equipping the organizations to sustain improvements over time, n = 263.
[3]n = 1,258.

McKinsey&Company


WITH GREAT WER COMES GREAT RESPOPONSBILIITY

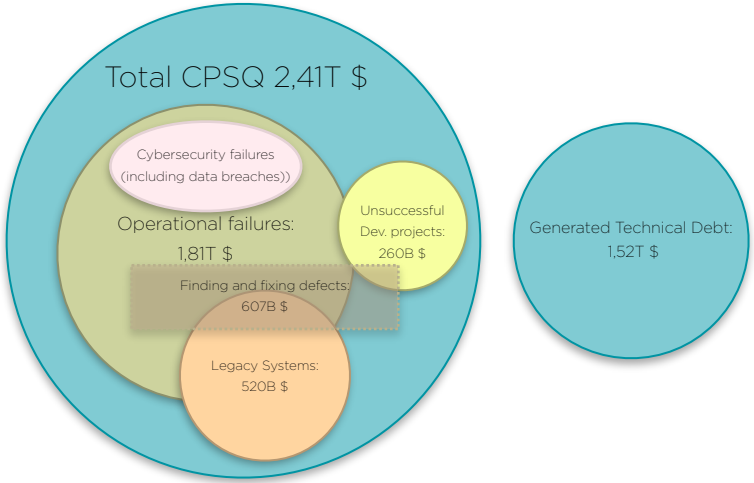
El software está habilitando la transformación digital de muchas organizaciones.

Pero al mismo tiempo, para otras se ha convertido en un verdadero quebradero de cabeza.

EL IMPACTO DEL SOFTWARE

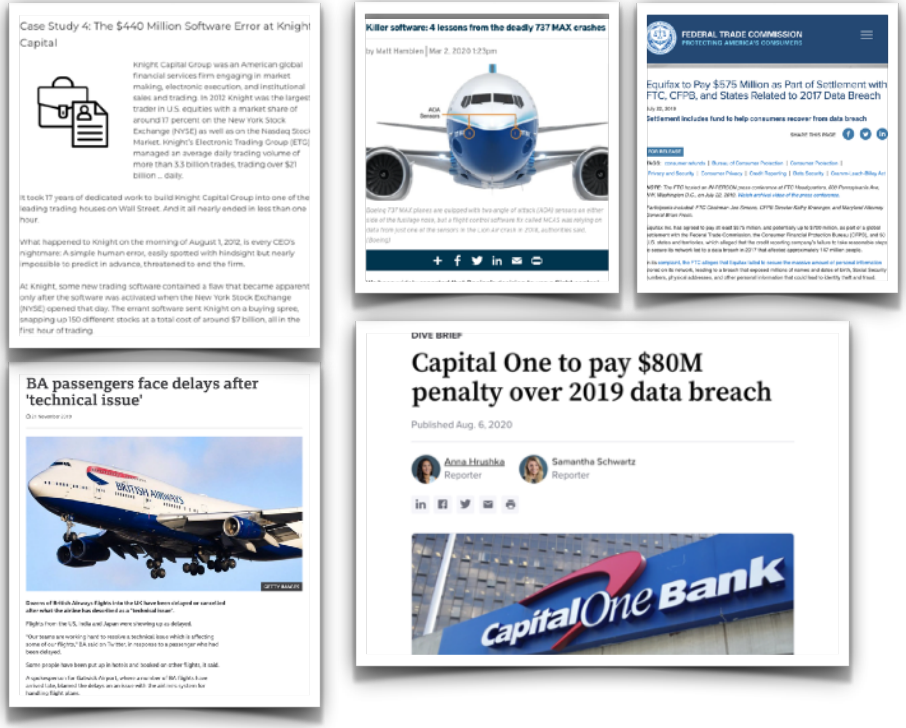
En el 2022 el coste económico que provocó el software de baja calidad en USA fue de 2,41T$ (PBI 25T$ izan zen)*.

Los problemas relacionados con la ciberseguridad coparon un porcentaje considerable.



* https://www.it-cisq.org/the-cost-of-poor-quality-software-in-the-us-a-2022-report/

EL IMPACTO DEL SOFTWARE DE BAJA CALIDAD



A pesar de que el conocimiento está más accesible que nunca, la situación no ha mejorado.

EL PROBLEMA DE DAR POR BUENO EL CÓDIGO QUE NOS ENCONTRAMOS EN INTERNET

BBC NEWS

Home | Israel-Gaza war | War in Ukraine | Climate | Video | World | UK | Business | Tech | Science

Tech

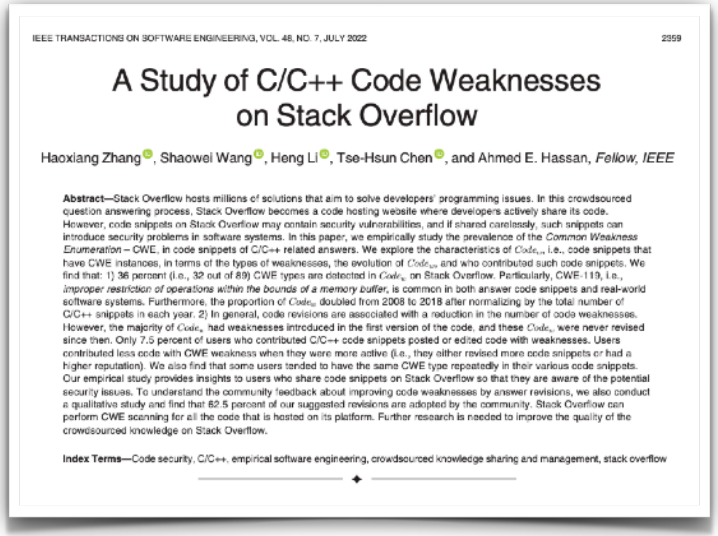
Copycat coders create 'vulnerable' apps

```
function use_unique(a) {
  for (var b = [], c = 0;c < a.length;c++) {
    0 == use_array(a[c], b) && b.push(a[c]);
  }
  return b.length;
}
function count_array_gen() {
  var a = 0, b = $("#User_logged").val(), b = b.rep
  inp_array = b.split(" ");
  input_sum = inp_array.length;
  for (var b = [], a = [], c = [], a = 0;a < inp_ar
    0 == use_array(inp_array[a], c) && (c.push(inp_
  }
  a = b;
  input_words = a.length;
```

Just copying code can make programs insecure, warn researchers

Lazy developers who copy solutions to tricky programming problems are creating apps that are vulnerable to attack, research suggests.

A team of computer scientists looked at more than 72,000 chunks of code found on the Stack Overflow website.



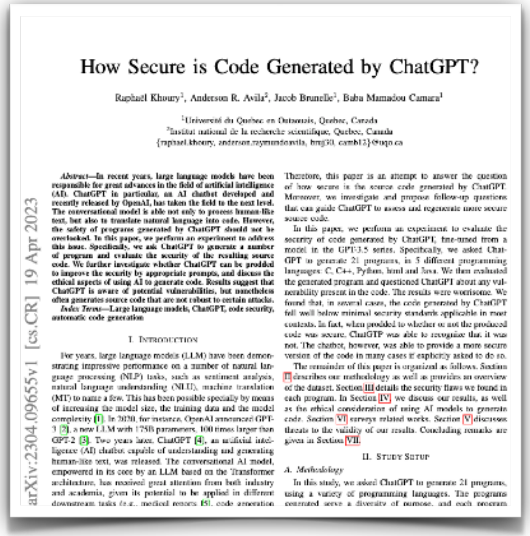
IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 48, NO. 7, JULY 2022 2359

A Study of C/C++ Code Weaknesses on Stack Overflow

Haoxiang Zhang, Shaowei Wang, Heng Li, Tse-Hsun Chen, and Ahmed E. Hassan, *Fellow, IEEE*

Abstract—Stack Overflow hosts millions of solutions that aim to solve developers' programming issues. In this crowdsourced question answering process, Stack Overflow becomes a code hosting website where developers actively share its code. However, code snippets on Stack Overflow may contain security vulnerabilities, and if shared carelessly, such snippets can introduce security problems in software systems. In this paper, we empirically study the prevalence of the *Common Weakness Enumeration* – CWE, in code snippets of C/C++ related answers. We explore the characteristics of $Code_{cs}$, i.e., code snippets that have CWE instances, in terms of the types of weaknesses, the evolution of $Code_{cs}$, and who contributed such code snippets. We find that: 1) 36 percent (i.e., 32 out of 89) CWE types are detected in $Code_{cs}$ on Stack Overflow. Particularly, CWE-119, i.e., *improper restriction of operations within the bounds of a memory buffer*, is common in both answer code snippets and real-world software systems. Furthermore, the proportion of $Code_{cs}$ doubled from 2008 to 2018 after normalizing by the total number of C/C++ snippets in each year. 2) In general, code revisions are associated with a reduction in the number of code weaknesses. However, the majority of $Code_{cs}$ had weaknesses introduced in the first version of the code, and these $Code_{cs}$ were never revised since then. Only 7.5 percent of users who contributed C/C++ code snippets posted or edited code with weaknesses. Users contributed less code with CWE weakness when they were more active (i.e., they either revised more code snippets or had a higher reputation). We also find that some users tended to have the same CWE type repeatedly in their various code snippets. Our empirical study provides insights to users who share code snippets on Stack Overflow so that they are aware of the potential security issues. To understand the community feedback about improving code weaknesses by answer revisions, we also conduct a qualitative study and find that 82.5 percent of our suggested revisions are adopted by the community. Stack Overflow can perform CWE scanning for all the code that is hosted on its platform. Further research is needed to improve the quality of the crowdsourced knowledge on Stack Overflow.

Index Terms—Code security, C/C++, empirical software engineering, crowdsourced knowledge sharing and management, stack overflow



How Secure is Code Generated by ChatGPT?

Raphaël Khoury[1], Anderson R. Avila[2], Jacob Brunelle[2], Baba Mamadou Camara[1]

[1]Université du Québec en Outaouais, Québec, Canada
[2]Institut national de la recherche scientifique, Québec, Canada
{raphael.khoury, anderson.raymundo.avila, brujl03, camb12}@uqo.ca

Abstract—In recent years, large language models have been responsible for great advances in the field of artificial intelligence (AI). ChatGPT in particular, an AI chatbot developed and recently released by OpenAI, has taken the field to the next level. The conversational model is able not only to process human-like text, but also to translate natural language into code. However, the safety of programs generated by ChatGPT should not be overlooked. In this paper, we perform an experiment to address this issue. Specifically, we ask ChatGPT to generate a number of program and evaluate the security of the resulting source code. We further investigate whether ChatGPT can be prodded to improve the security by appropriate prompts, and discuss the ethical aspects of using AI to generate code. Results suggest that ChatGPT is aware of potential vulnerabilities, but nonetheless often generates source code that are not robust to certain attacks.

Index Terms—Large language models, ChatGPT, code security, automatic code generation

I. Introduction

For years, large language models (LLM) have been demonstrating impressive performance on a number of natural language processing (NLP) tasks, such as sentiment analysis, natural language understanding (NLU), machine translation (MT) to name a few. This has been possible specially by means of increasing the model size, the training data and the model complexity [1]. In 2020, for instance, OpenAI announced GPT-3 [2], a new LLM with 175B parameters, 100 times larger than GPT-2 [3]. Two years later, ChatGPT [4], an artificial intelligence (AI) chatbot, capable of understanding and generating human-like text, was released. The conversational AI model, empowered in its core by an LLM based on the Transformer architecture, has received great attention from both industry and academia, given its potential to be applied in different downstream tasks (e.g., medical reports [5], code generation

Therefore, this paper is an attempt to answer the question of how secure is the source code generated by ChatGPT. Moreover, we investigate and propose follow-up questions that can guide ChatGPT to assess and regenerate more secure source code.

In this paper, we perform an experiment to evaluate the security of code generated by ChatGPT, fine-tuned from a model in the GPT-3.5 series. Specifically, we asked ChatGPT to generate 21 programs, in 5 different programming languages: C, C++, Python, html and Java. We then evaluated the generated program and questioned ChatGPT about any vulnerability present in the code. The results were worrisome. We found that, in several cases, the code generated by ChatGPT fell well below minimal security standards applicable in most contexts. In fact, when prodded to whether or not the produced code was secure, ChatGPT was able to recognize that it was not. The chatbot, however, was able to provide a more secure version of the code in many cases if explicitly asked to do so.

The remainder of this paper is organized as follows. Section II describes our methodology as well as provides an overview of the dataset. Section III details the security flaws we found in each program. In Section IV, we discuss our results, as well as the ethical consideration of using AI models to generate code. Section VI surveys related works. Section V discusses threats to the validity of our results. Concluding remarks are given in Section VII.

II. Study Setup

A. Methodology

In this study, we asked ChatGPT to generate 21 programs, using a variety of programming languages. The programs contrasted across a diversity of purposes, and each program

Es imprescindible que los desarrolladores tengan la seguridad en mente en todo el ciclo de vida del código.

La ciberseguridad abarca un espectro muy amplio de aspectos.

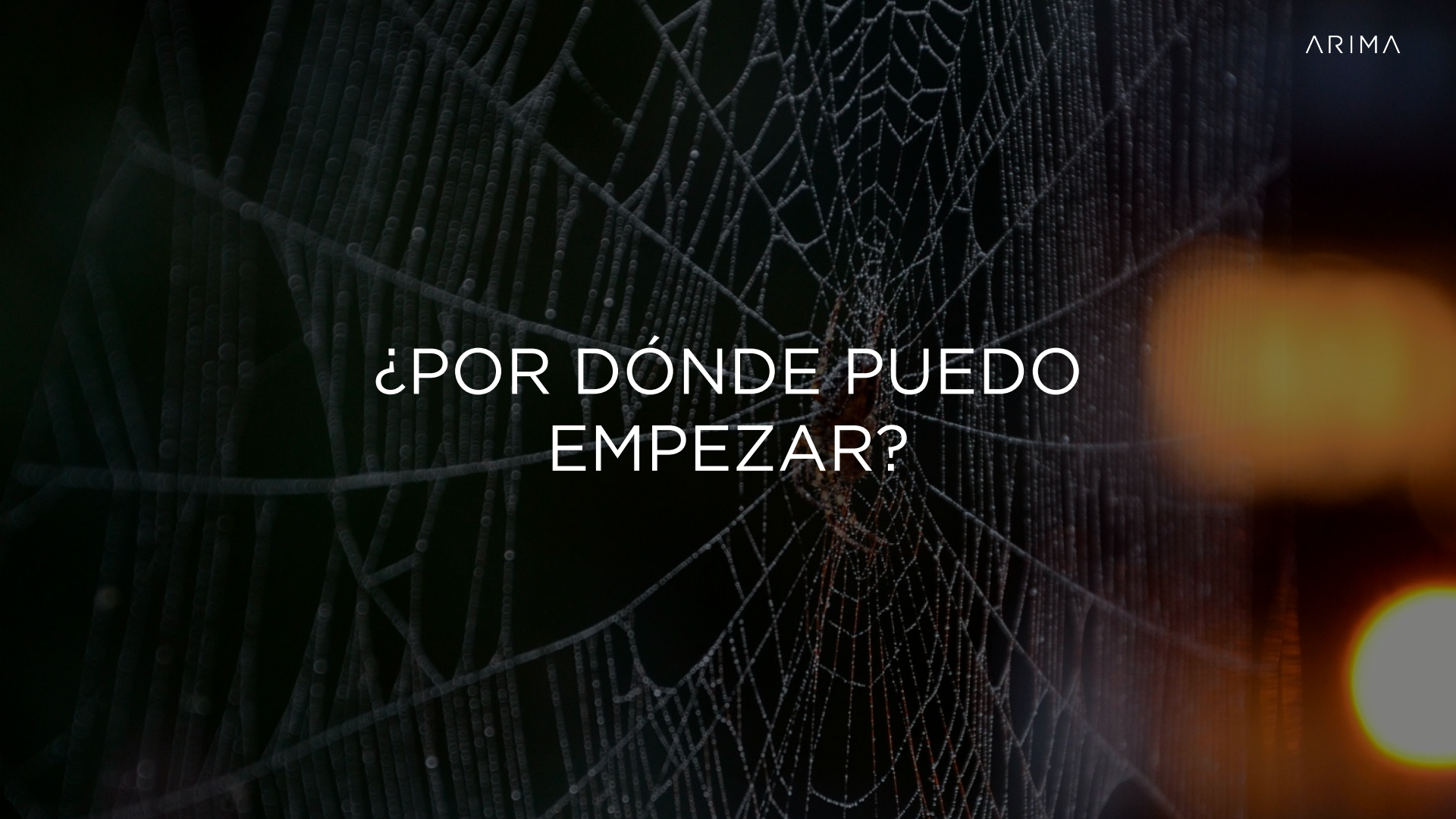
En esta formación nos vamos a centrar en el desarrollo de software seguro en entornos web.

¿POR QUÉ WEB?

Es algo que está muy accesible para todos: Web apps, APIs, etc.

La superficie de ataque es infinita y muy fácil de explotar


ARIMA
I'M A HACKER



¿POR DÓNDE PUEDO EMPEZAR?

ARIMA

02./ OWASP

OWASP
(Open Worldwide Application Security Project)

OWASP

La OWASP es una organización sin ánimo de lucro que trabaja para mejorar la seguridad del software.

Tiene miles de miembros, imparten conferencias y hacen campañas de divulgación y concienciación.

Todo el material, herramientas y proyectos que desarrollan son libres y gratuitas.

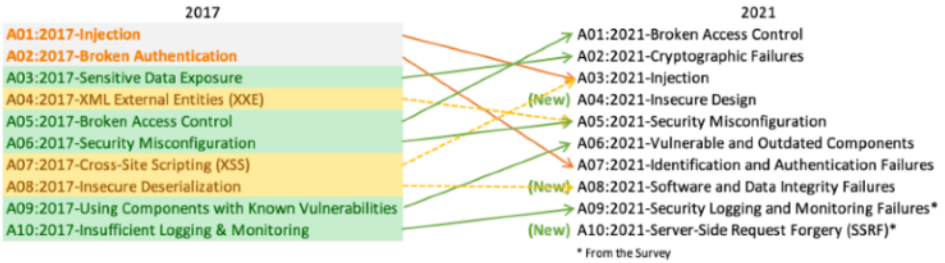
TOP 10

OWASP TOP 10

La OWASP tiene un ranking en el que recoge las 10 vulnerabilidades críticas más comunes en aplicaciones web.

Es una lista que se actualiza cada cierto tiempo en base a la información recogida desde varias organizaciones relacionadas con la ciberseguridad y la comunidad.

En esta formación vamos a apoyarnos en este ranking para explotar algunas de estas vulnerabilidades.



LET ME INTRODUCE YOU OUR VICTIM

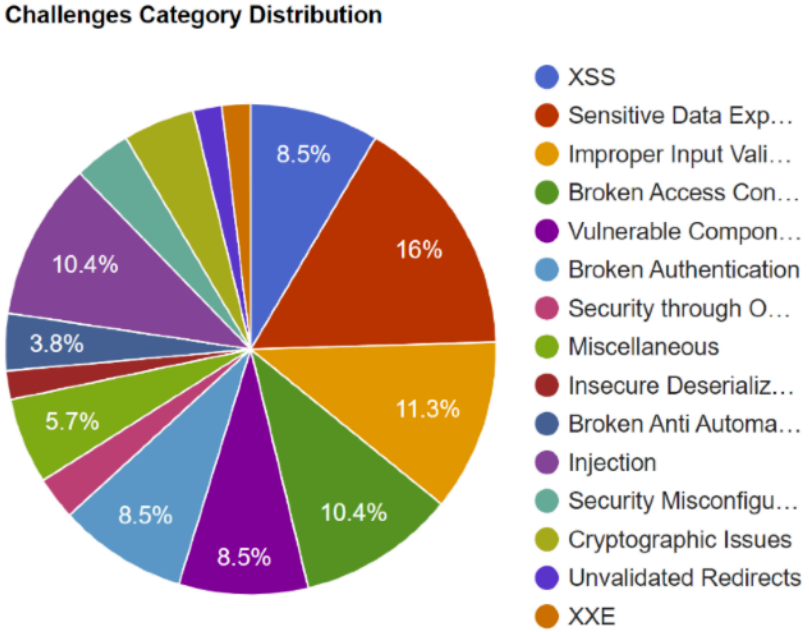
Juice Shop es una aplicación llena de vulnerabilidades

JUICE SHOP

"Juice Shop" es una aplicación web open source que está llena de vulnerabilidades.

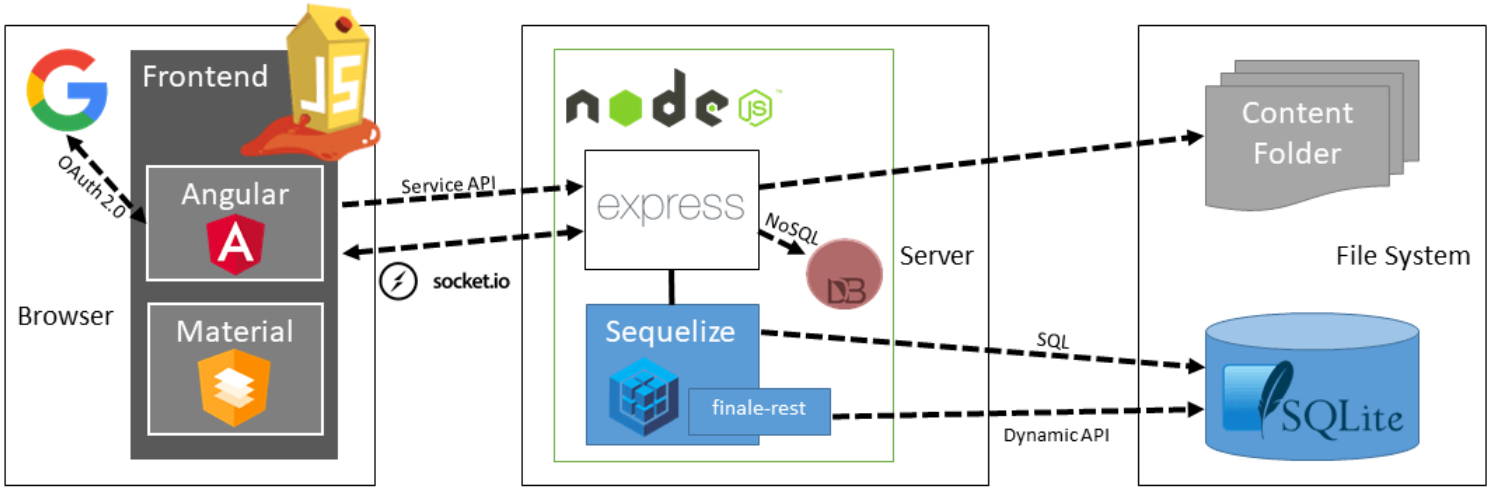
El objetivo es encontrar y explotar estas vulnerabilidades.

A medida que exploremos cada una de las vulnerabilidades, repasaremos por qué suelen ocurrir y qué podemos hacer para mitigar los riesgos.

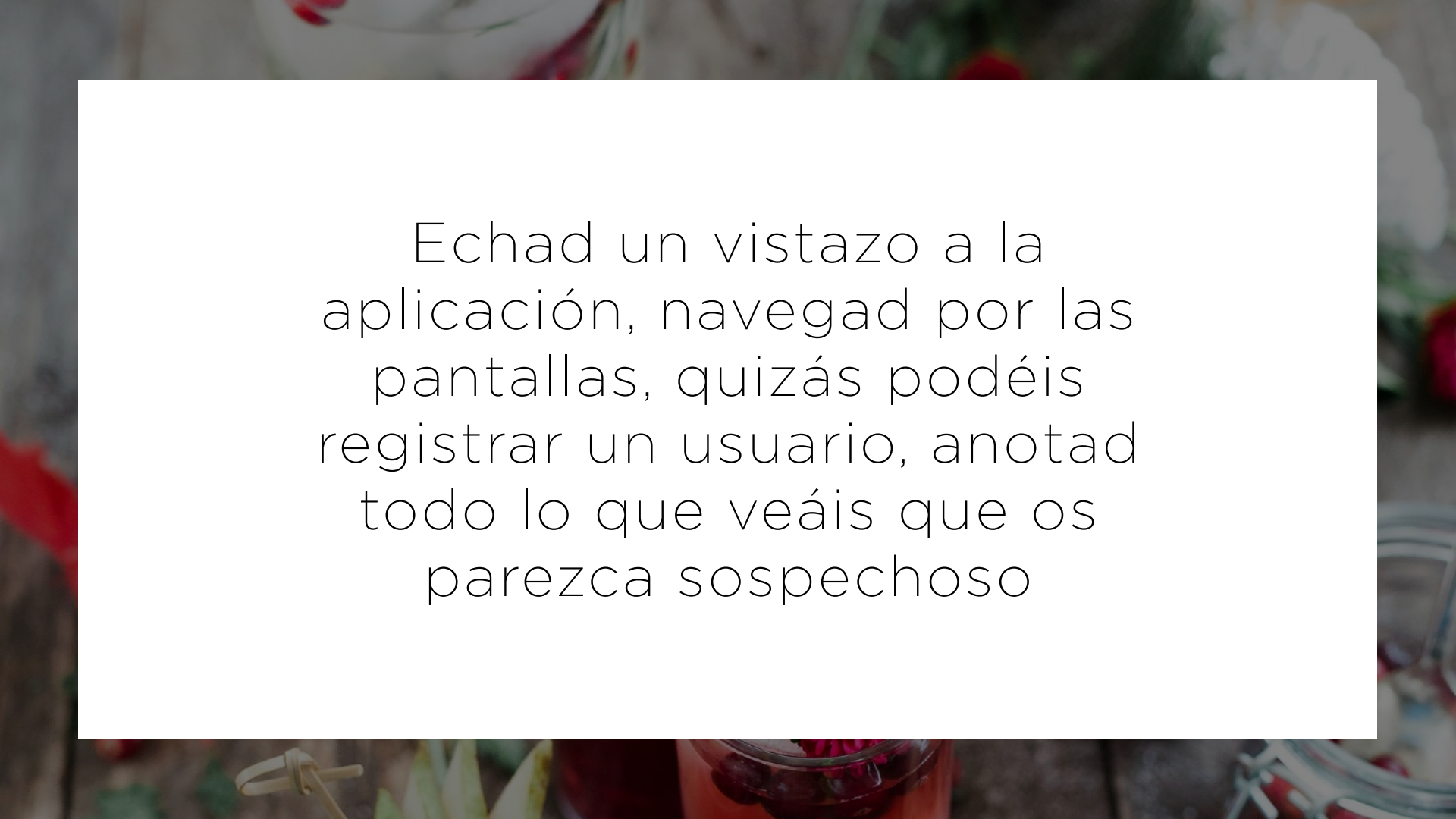


¿Desde el punto de vista de su arquitectura, qué clasificación de tipos de aplicaciones web haríais?

ARQUITECTURA DE LA APLICACIÓN QUE VAMOS A UTILIZAR



JUICE SHOP URL
http://192.168.214.**[202-230]**



Echad un vistazo a la aplicación, navegad por las pantallas, quizás podéis registrar un usuario, anotad todo lo que veáis que os parezca sospechoso

ARIMA

03./ LETS HACK!

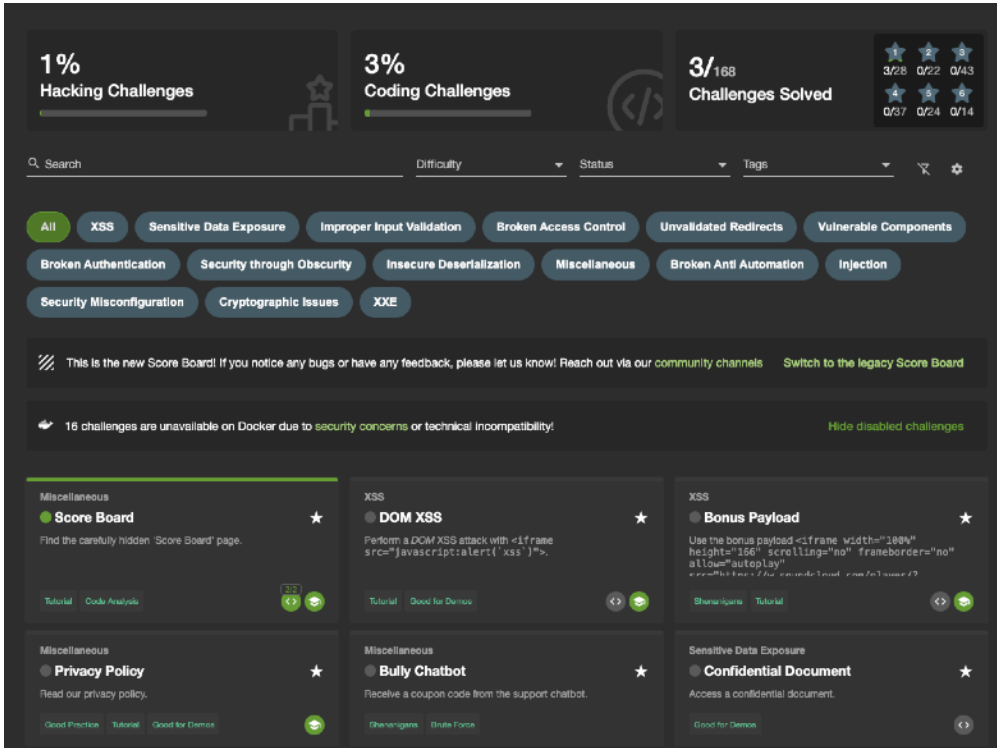
CALLENGE I:
FIND THE SCORE BOARD

FIND THE SCORE BOARD

La aplicación tiene una tabla de puntuación oculta.

En esta tabla se muestran todos los retos disponibles y se van puntuando aquellos que desbloqueamos.

¡A ver si sois capaces de encontrarla!




LET’S HACK!

CALLENGE II:
XSS

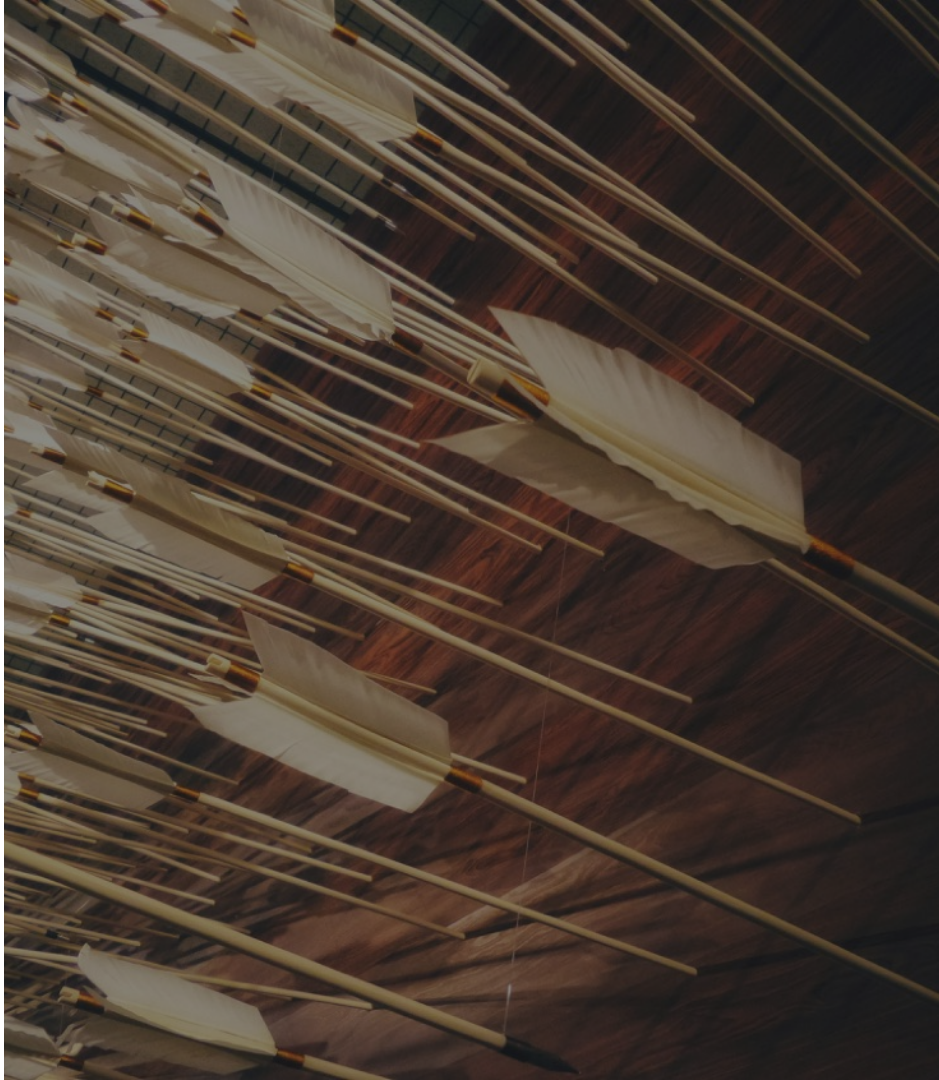
XSS
A3:2021 INJECTION

Los ataques de inyección en general ocurren cuando no se filtra, valida y sanitiza información provista por los usuarios de una aplicación y se utiliza directamente para ejecutar acciones.

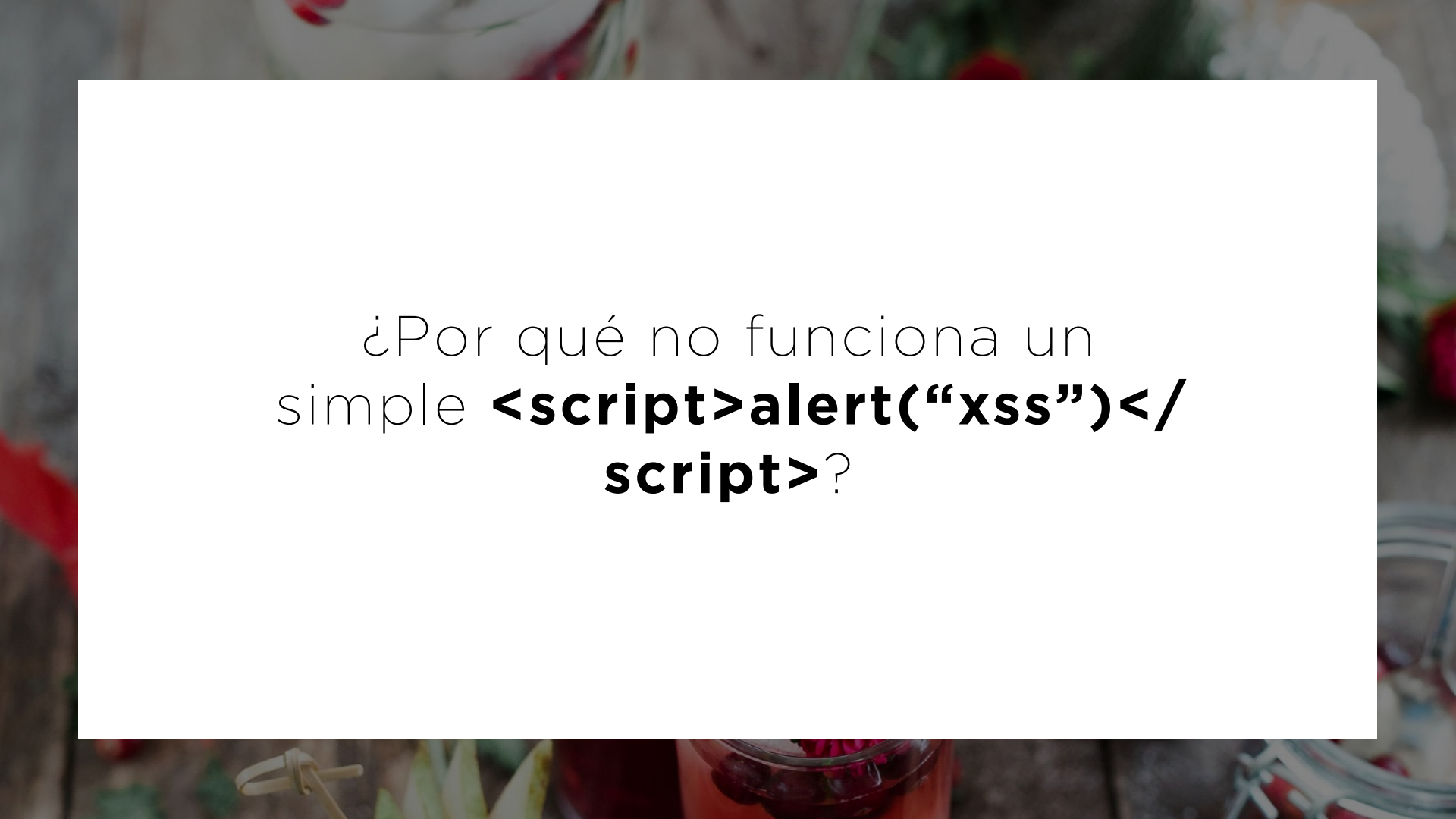
Habitualmente esto provoca que el usuario pueda ejecutar código de manera arbitraria.

Mediante DOM XSS, un usuario puede provocar que otro usuario ejecute código malicioso simplemente haciéndole llegar un link que incluye el código que se va a inyectar en el DOM.

¿Cómo podemos explotar esta vulnerabilidad en Juice Shop?

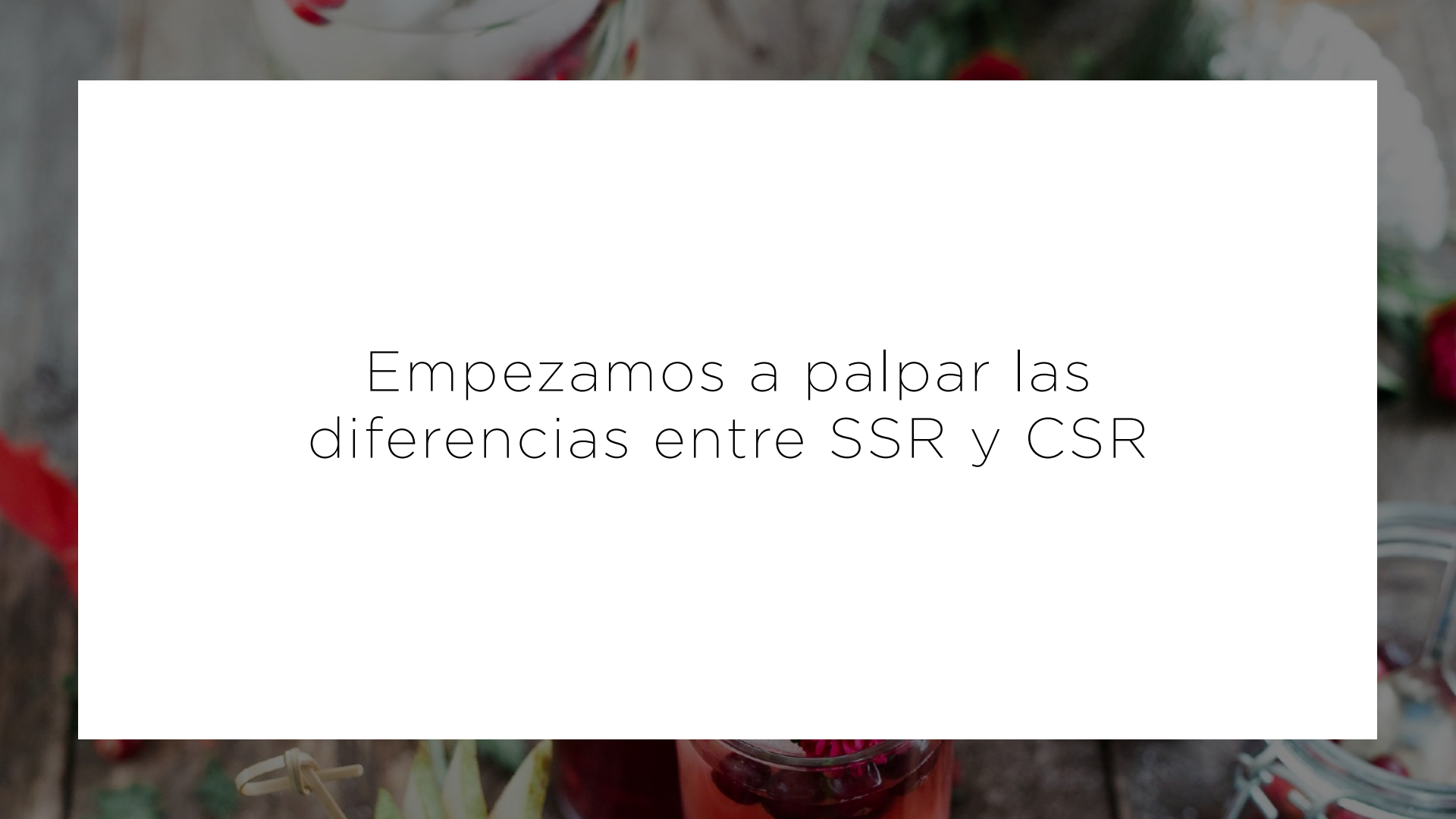



LET’S HACK!



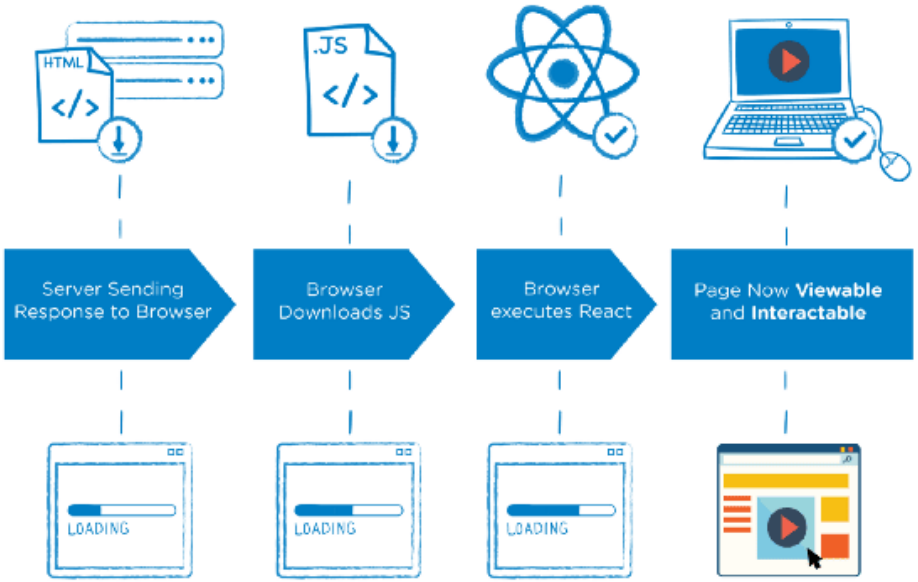
¿Por qué no funciona un simple **<script>alert(“xss”)</script>**?

Vamos a probarlo en
https://xss-game.appspot.com/

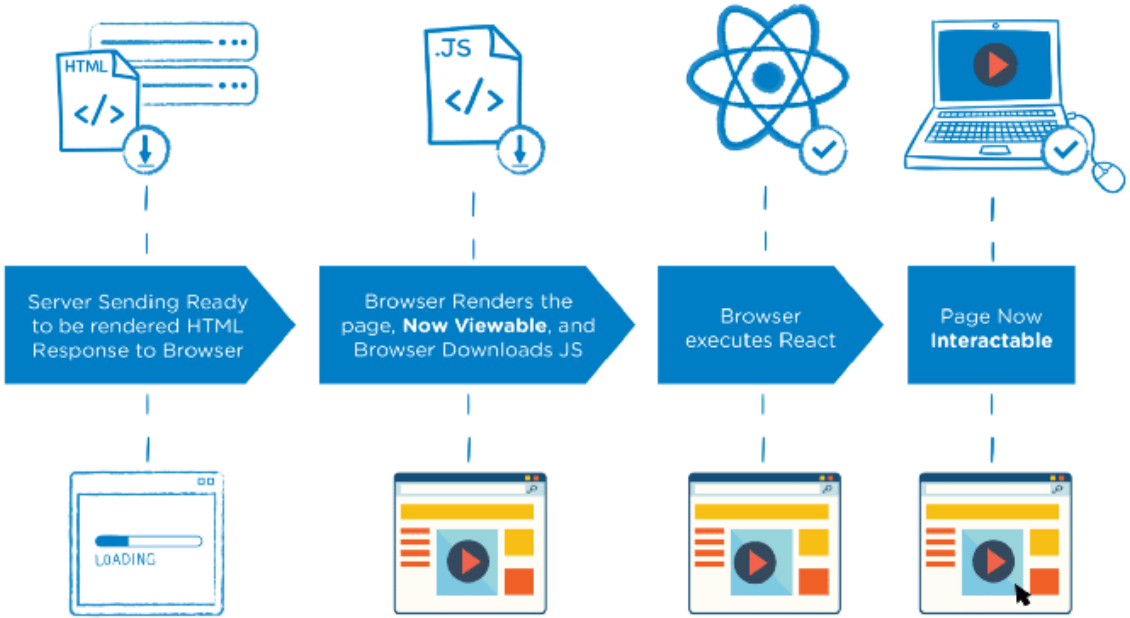


Empezamos a palpar las diferencias entre SSR y CSR

CSR


HTML
.JS
Server Sending Response to Browser
Browser Downloads JS
Browser executes React
Page Now **Viewable** and **Interactable**
LOADING
LOADING
LOADING

SSR


HTML
.JS
Server Sending Ready to be rendered HTML Response to Browser
Browser Renders the page, **Now Viewable**, and Browser Downloads JS
Browser executes React
Page Now **Interactable**
LOADING

XSS
SSR VS CSR

Si vemos el código fuente de las dos aplicaciones que renderizan el HTML en el cliente, podemos ver que ambas acaban utilizando la propiedad innerHTML:

- La directiva Angular [innerHTML] en Juice Shop.
- containerEl.innerHTML += html; en xss-gamespot.

InnerHTML no ejecuta las etiquetas script.

Security considerations

It is not uncommon to see `innerHTML` used to insert text into a web page. There is potential for this to become an attack vector on a site, creating a potential security risk.

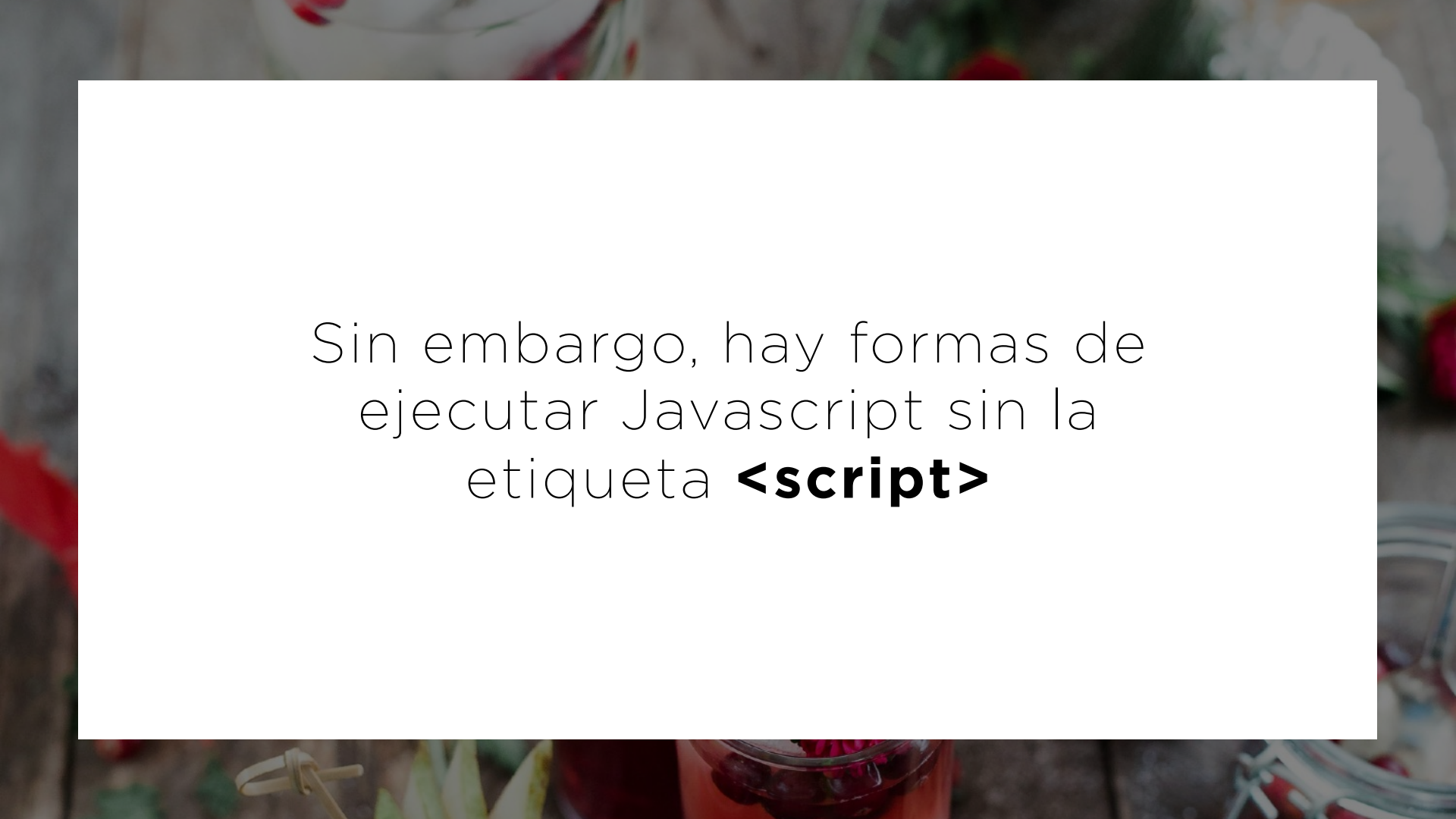
JS

```
let name = "John";
// assuming 'el' is an HTML DOM element
el.innerHTML = name; // harmless in this case

// …

name = "<script>alert('I am John in an annoying alert!')</script>";
el.innerHTML = name; // harmless in this case
```

Although this may look like a cross-site scripting attack, the result is harmless. HTML specifies that a `<script>` tag inserted with `innerHTML` should not execute.



Sin embargo, hay formas de ejecutar Javascript sin la etiqueta **<script>**

XSS PAYLOADS

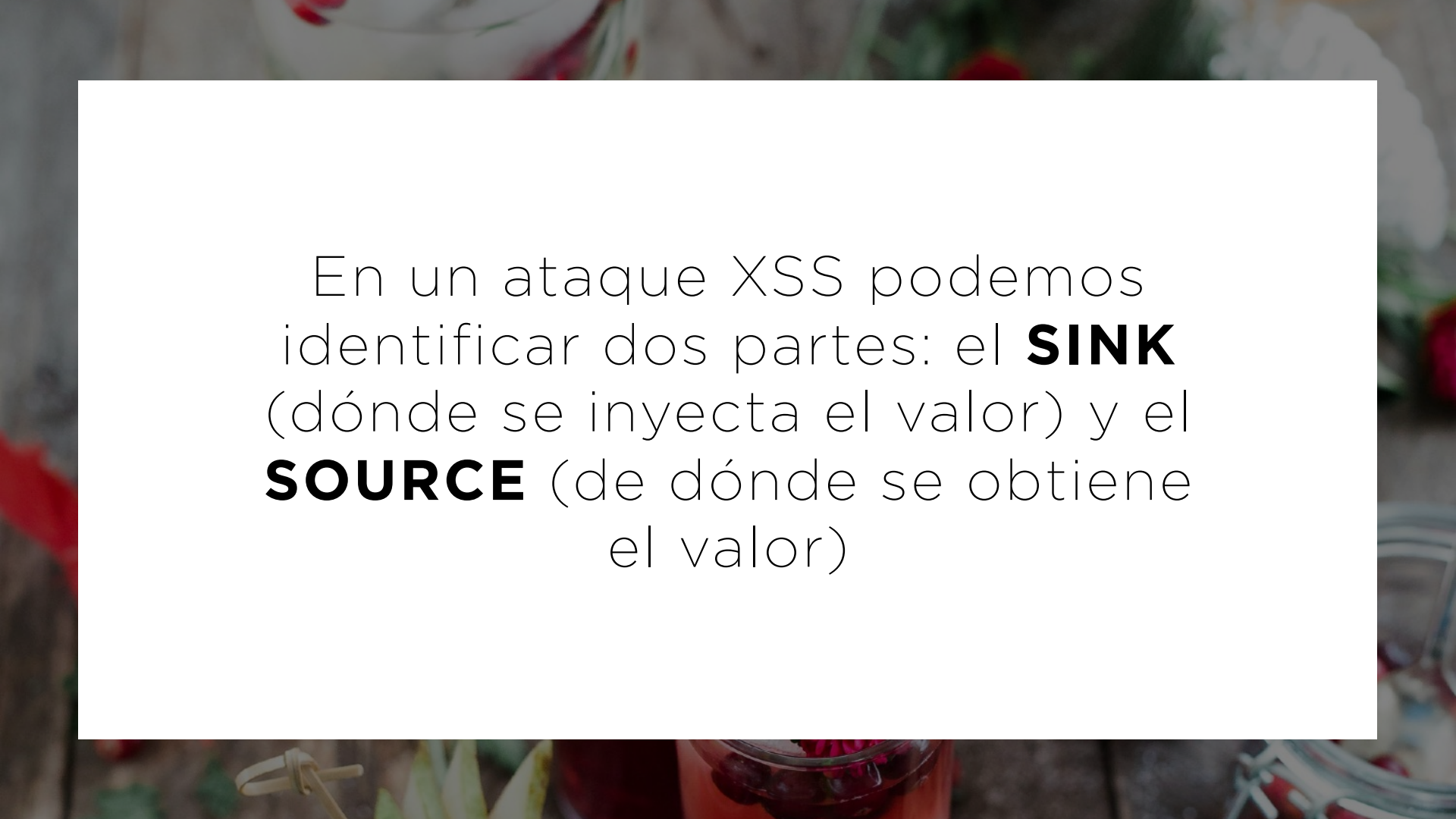
Podemos utilizar cualquier etiqueta que tenga la posibilidad de ejecutar un callback cuando ocurre un evento que podemos provocar (URLs que no existen, pintado de la etiqueta, etc.). Algunos ejemplos:

/ <img src="x" onerror="">

/ <audio src="x" onerror="">

/ <video src="x" onerror="">

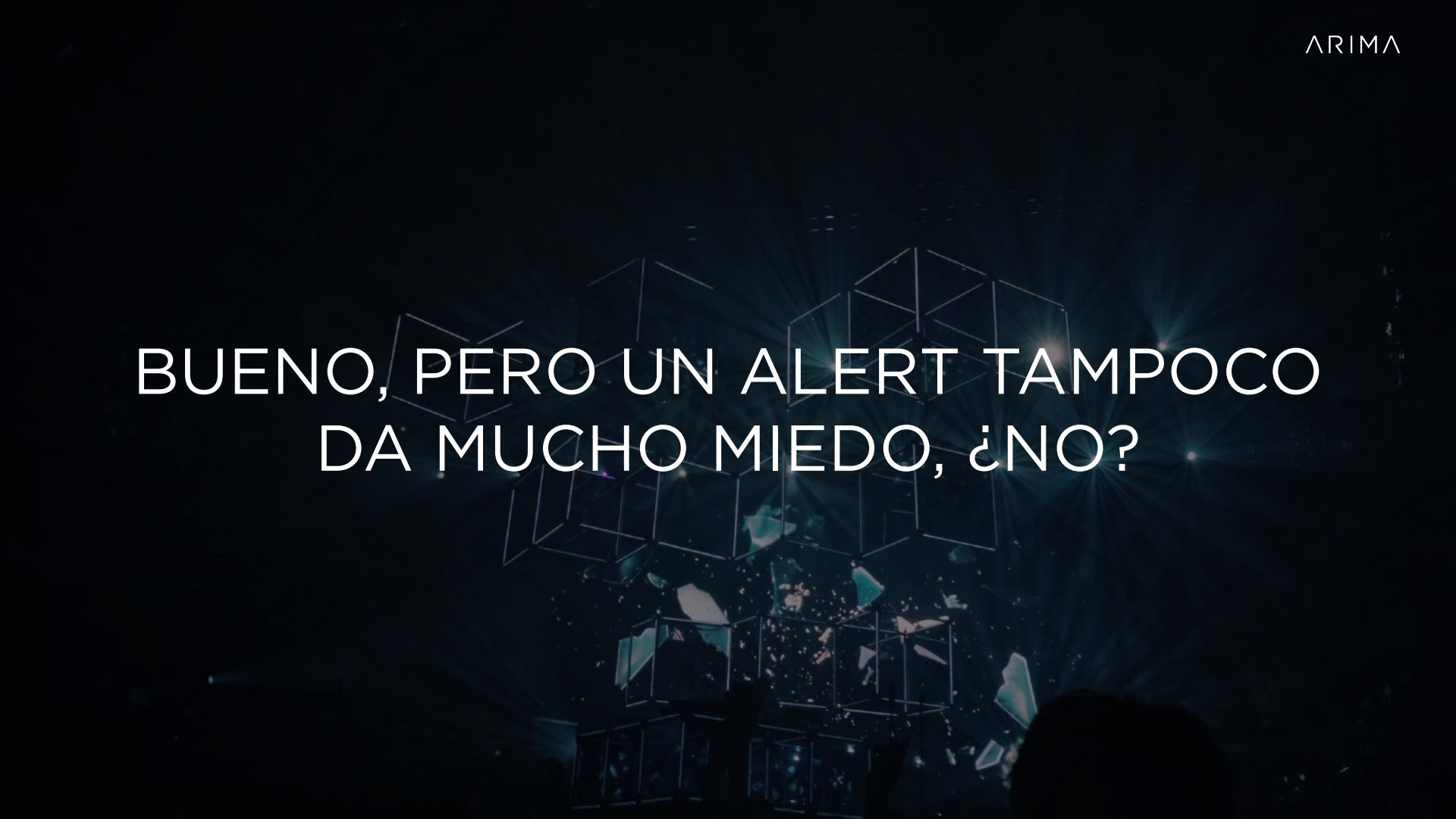
* https://portswigger.net/web-security/cross-site-scripting/cheat-sheet



En un ataque XSS podemos identificar dos partes: el **SINK** (dónde se inyecta el valor) y el **SOURCE** (de dónde se obtiene el valor)

A veces encontrar el sink no es tan evidente https://xss-game.appspot.com/level3

Vamos a ver cómo podemos explotar otro caso en el servidor https://xss-game.appspot.com/level4


BUENO, PERO UN ALERT TAMPOCO
DA MUCHO MIEDO, ¿NO?

MYSPACE “SAMY” WORM (2005)

Samy Kamkar inyectó código Javascript en su propio perfil de forma que cuando otros lo veían, automáticamente les añadía como amigo y promocionaban su perfil.

En cuestión de horas, millones de perfiles fueron infectados provocando que el rendimiento se degradara considerablemente. Tuvieron que parar la plataforma para corregir el problema.

TWITTER WORM (2009)

Un hacker de 17 años aprovechó una vulnerabilidad de stored XSS que encontró en Twitter para ejecutar un payload que se aprovechavaba del evento “onmouseover” de los links. Cuando los usuarios hacían “hover“ sobre estos links, se hacía un retweet automáticamente de los mismos provocando que se viralizasen.

Afectó a numerosos usuarios de Twitter y provocó un problema de reputación a la red social.

NEWS

Watch ONE-MINUTE WORLD NEWS

News Front Page

Africa
Americas
Asia-Pacific
Europe
Middle East
South Asia
UK
Business
Health
Science & Environment
Technology
Entertainment
Also in the news

Video and Audio

Programmes
Have Your Say
In Pictures
Country Profiles
Special Reports

Related BBC sites
Sport
Weather
On This Day
Editors' Blog
BBC World Service

Page last updated at 08:36 GMT, Tuesday, 14 April 2009 09:36 UK

E-mail this to a friend Printable version

Twitter all clear after worm wave

Twitter has been given the all clear after a worm infected "tens of thousands of users". But experts say the attack could have been much worse.

Over the weekend, a self-replicating computer program, or worm, began to infect profiles on the social network.

Twitter was hit by a worm over the weekend

The worm was set up to promote a Twitter rival site, showing unwanted messages on infected user accounts.

Michael Mooney, a 17-year-old US student, told the Associated Press he created the worm to promote his site.

Mooney, who lives in Brooklyn, New York, said he wanted to expose vulnerabilities in Twitter. He told AP: "I really didn't think it was going to get that much attention, but then I started to see all these stories about it and thought, 'Oh, my God'."

The worm worked by encouraging users to click on a link to the rival Twitter site, called StalkDaily.com.

Once the link was clicked, infected users themselves automatically began to send out messages to friends, promoting the site.

No personal or sensitive information, such as passwords, was compromised in the attacks, according to Twitter, which has more than seven million users.

WHATSAPP WEB XSS (2020)

Un investigador encontró una vulnerabilidad XSS en el cliente web de WhatsApp. El atacante podía aprovecharse de esta vulnerabilidad para enviar mensaje especialmente preparado para contener código malicioso que se ejecutaría cuando el recipiente viera el mensaje.

Esta vulnerabilidad podría haber permitido a los atacantes a comprometer los datos del usuario y potencialmente, hacerse con el control de sus cuentas.

Vulnerability in WhatsApp Desktop Exposed User Files

Facebook has patched a vulnerability in WhatsApp Desktop that could allow an attacker to launch cross-site scripting (XSS) attacks and access files from the victim's system when paired with WhatsApp for iPhone.

By Ionut Arghire
February 5, 2020

Facebook has patched a vulnerability in WhatsApp Desktop that could allow an attacker to launch cross-site scripting (XSS) attacks and access files from the victim's system when paired with WhatsApp for iPhone.

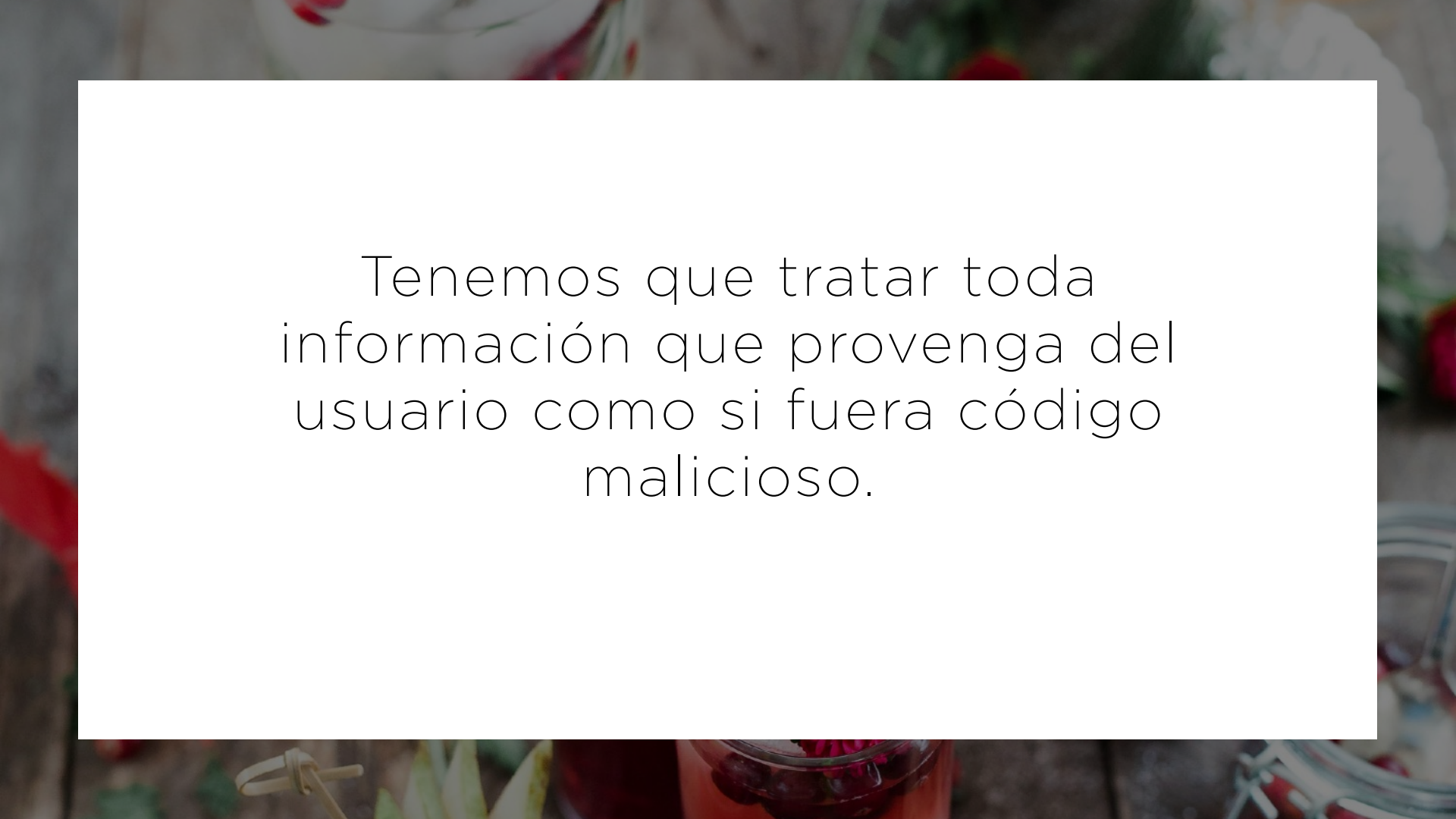
Tracked as CVE-2019-18426 and considered high severity (CVSS score 8.2), the security bug could be exploited by sending to the victim a specially crafted text message that included a link preview. Both Windows and macOS users were impacted.

The vulnerability was discovered by PerimeterX security researcher Gal Weizman, who said he found multiple issues in WhatsApp Desktop, starting with an open redirect into persistent XSS and Content Security Policy (CSP) bypass, and then a "cross platforms read from the local file system."

What the security researcher found was that he could bypass WhatsApp's CSP to execute code on a target system using maliciously crafted messages.

One of the main issues Weizman identified was that an attacker could modify WhatsApp reply messages to include quotes of messages the recipient never sent.

¿CÓMO NOS PODEMOS PROTEGER?



Tenemos que tratar toda información que provenga del usuario como si fuera código malicioso.

SAFE SINKS

A la hora de inyecta código al HTML, hay que utilizar atributos o métodos que se encargan de sanitizar los valores. Por ejemplo, utilizar textContent en lugar de innerHTML.

Los frameworks codifican por defecto, tienen métodos especiales con nombres muy explícitos para casos legítimos:

- **dangerouslySetInnerHTML** en React.
- **bypassSecurityTrustAs*** en Angular.
- **{% autoescape false %}** en Flask (Python).

```
elem.textContent = dangerVariable;
elem.insertAdjacentText(dangerVariable);
elem.className = dangerVariable;
elem.setAttribute(safeName, dangerVariable);
formfield.value = dangerVariable;
document.createTextNode(dangerVariable);
document.createElement(dangerVariable);
elem.innerHTML = DOMPurify.sanitize(dangerVar);
```

```
&    &amp;
<    &lt;
>    &gt;
"    &quot;
'    &#x27;
```

Utilizar unos Content Security Policy (CSP) headers lo más estrictos posibles para minimizar los posibles daños.

CSP

Son unas cabeceras específicas para establecer qué recursos se pueden cargar desde un dominio determinado.

De esta forma, el navegador únicamente ejecutará scripts que provengan de estos dominios ignorando el resto (incluidos los scripts en línea o los callbacks definidos en las etiquetas HTML).

* https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

Algunos ejemplos

Solo se permite contenido que proviene del mismo origen (excluyendo subdominios).

```HTTP
Content-Security-Policy: default-src 'self'
```

Permite contenido de un dominio de confianza y de todos sus subdominios.

```HTTP
Content-Security-Policy: default-src 'self' example.com *.example.com
```

Permite incluir imágenes de cualquier dominio, restringe el audio y el vídeo a unos proveedores de confianza, y los scripts de un host.

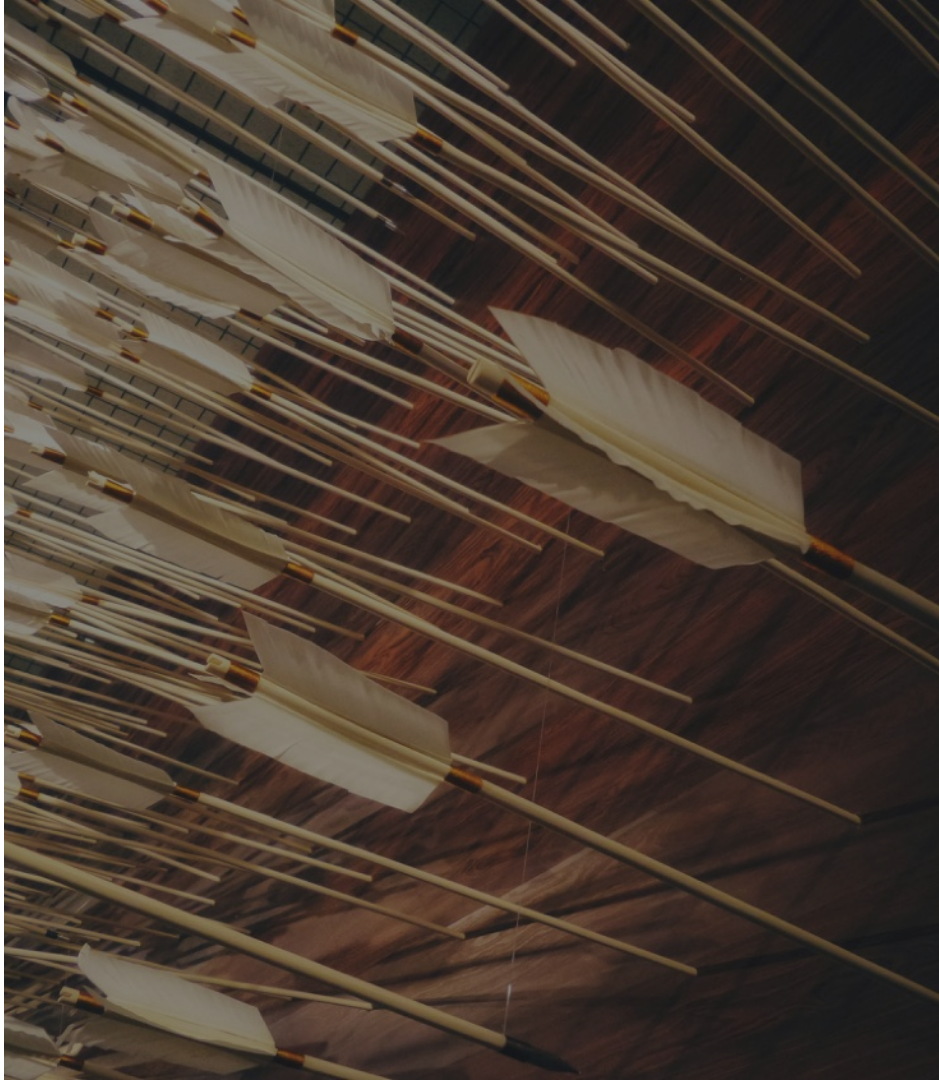
```HTTP
Content-Security-Policy: default-src 'self'; img-src *; media-src example.org example.net; script-src userscripts.example.com
```

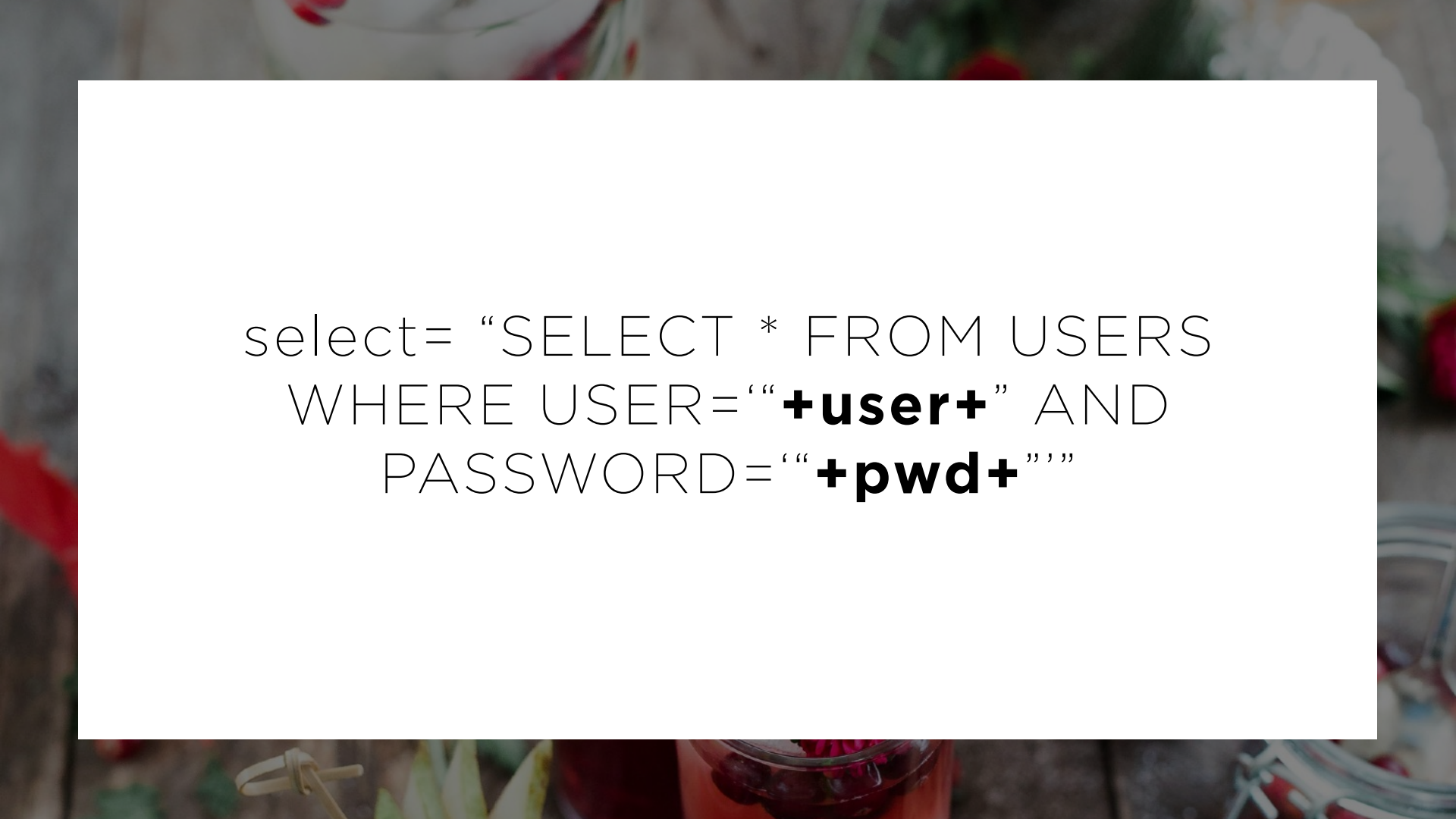
CALLENGE III:
SQL INJECTION

SQLI
A3:2021 INJECTION

Como en el resto de ataques de inyección, el problema se da cuando se concatena un valor provisto por el cliente directamente en una query sin “sanitizar”.

Esto abre la posibilidad a que un atacante pueda manipular la query y ejecutar consultas arbitrarias.




select= “SELECT * FROM USERS WHERE USER=‘“+user+” AND PASSWORD=‘“+pwd+”’”

Vamos a explotarlo en Juice Shop

ALGUNOS CASOS POPULARES

FREEPIK

En el 2020 Freepik uno de los sitios web de recursos gráficos más grandes del mundo con 18 millones de usuarios únicos mensuales sufrió un ataque SQL Injection.

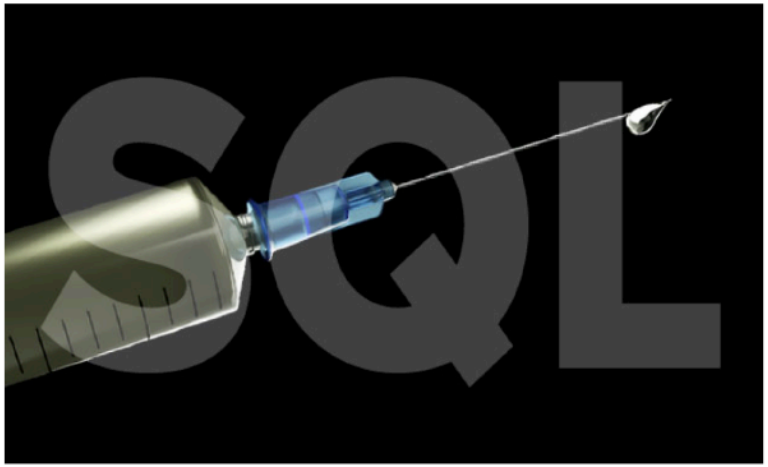
Los atacantes pudieron robar emails y passwords haseados de 8,3 millones de usuarios.

Massive Freepik Data Breach Tied to SQL Injection Attack

Millions of Email Addresses, Hashed Passwords Leaked

Doug Olenick (DougOlenick) • August 24, 2020

Share Tweet Share

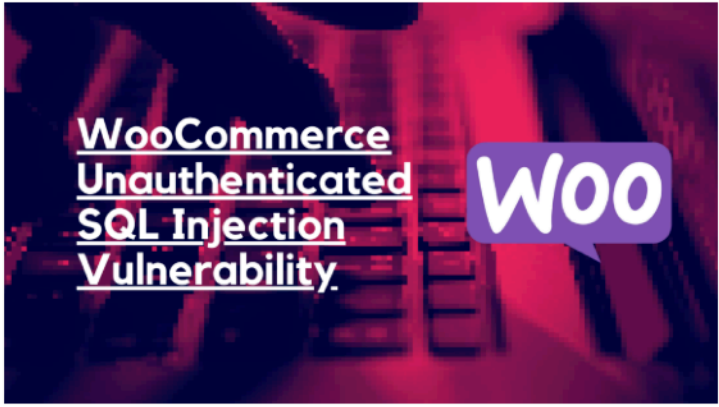


Freepik Co. says an SQL injection attack led to the leak of 8.3 million email addresses and 3.7 million hashed passwords for users of its Freepik graphic resources app and Flaticon icon database platform.

WOOCOMMERCE

En 2021, se descubrió que el popular plugin de ecommerce para el CMS Wordpress WooCommerce, era vulnerable a SQLi.

Se vieron afectados más de 5 millones de sitios web.

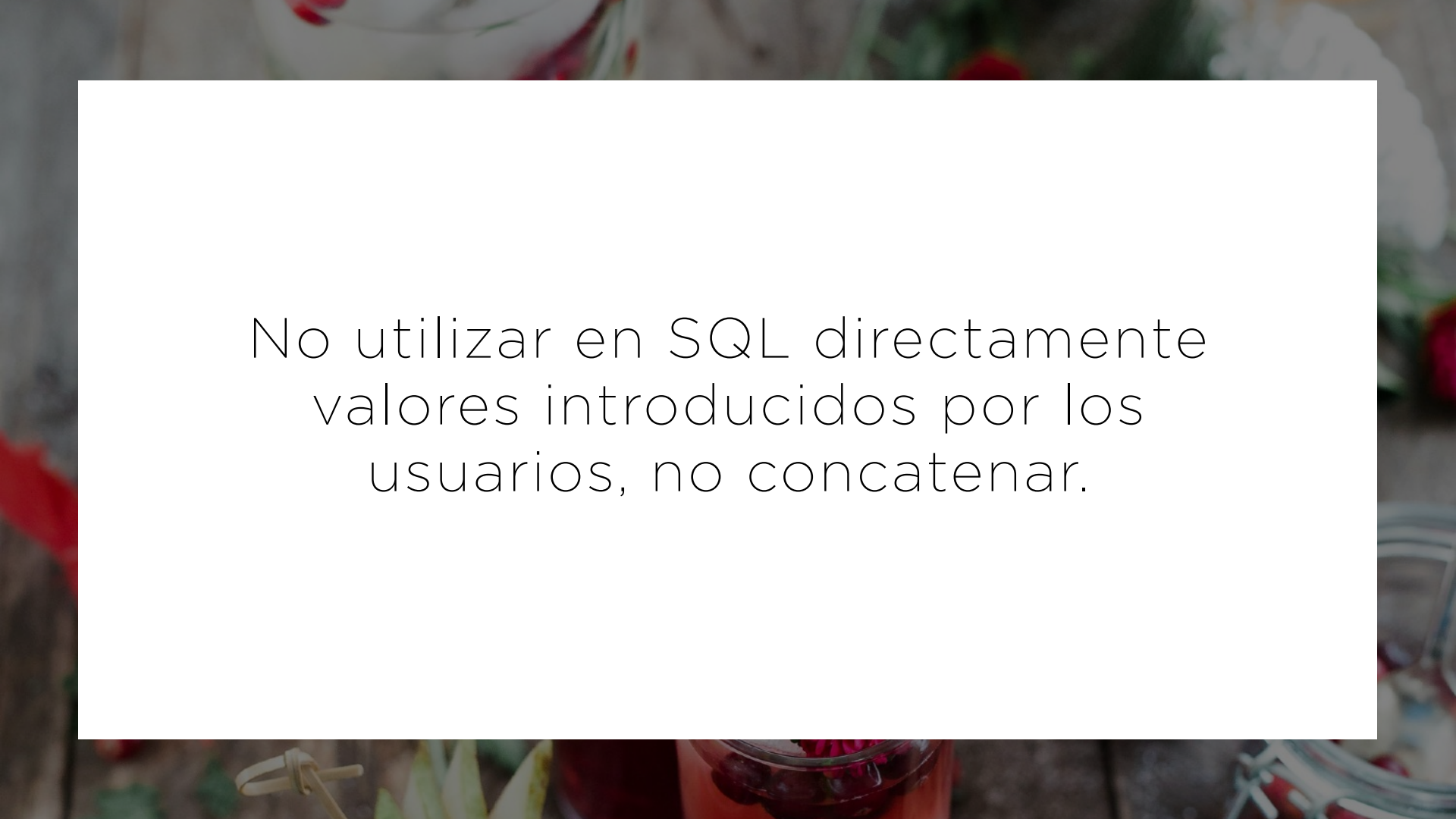


WooCommerce Unauthenticated SQL Injection Vulnerability

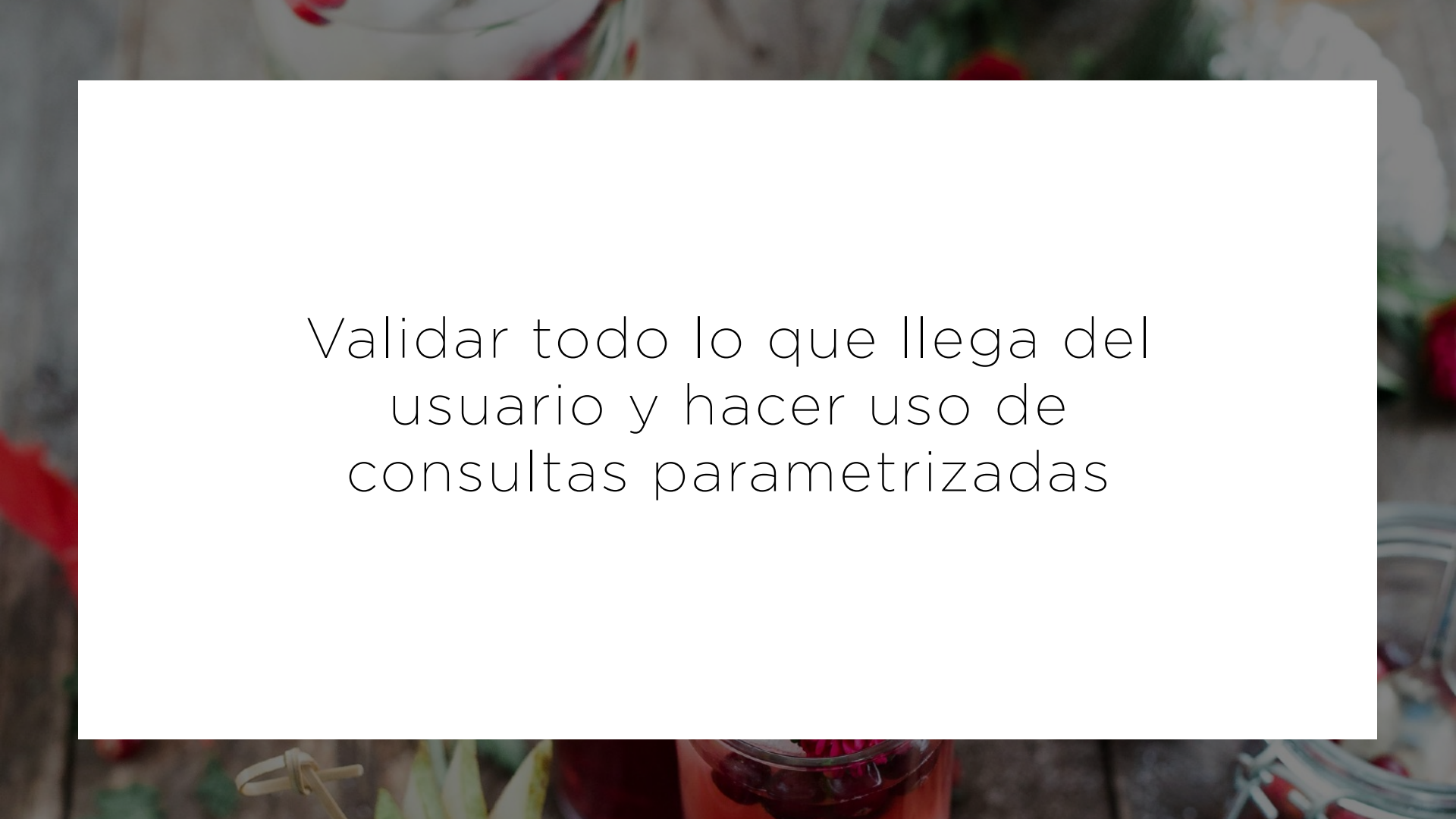
2021-07-19

On 15th July 2021, news was going around regarding an unauthenticated SQL Injection in WooCommerce. WooCommerce released a blog post about the vulnerabilities here: https://woocommerce.com/posts/critical-vulnerability-detected-july-2021/#. The vulnerabilities were detected on the 13th of July and fixed in WooCommerce versions 3.3.6 to 5.5.1 and WooCommerce Blocks versions 2.5.16 to 5.5.1. This blog post is a short analysis of the vulnerabilities, the patch, and then a PoC for the bugs!

¿CÓMO NOS PODEMOS PROTEGER?



No utilizar en SQL directamente valores introducidos por los usuarios, no concatenar.



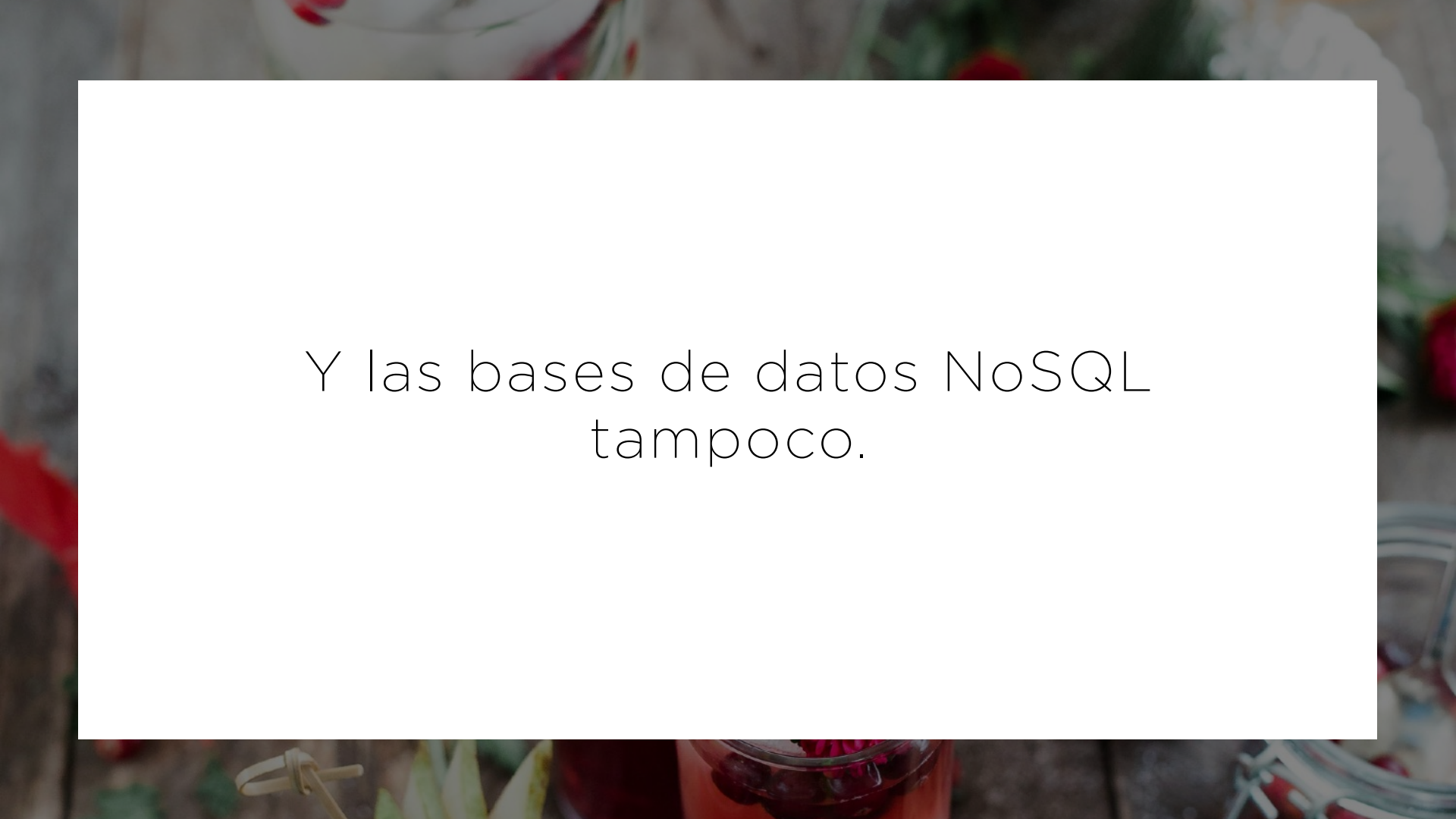
Validar todo lo que llega del usuario y hacer uso de consultas parametrizadas

JAVA PreparedStatement

```
// This should REALLY be validated too
String custname = request.getParameter("customerName");
// Perform input validation to detect attacks
String query = "SELECT account_balance FROM user_data WHERE user_name = ?";
PreparedStatement pstmt = connection.prepareStatement(query);
pstmt.setString(1, custname);
ResultSet results = pstmt.executeQuery();
```

Todos los frameworks de cualquier lenguaje tienen este concepto.

Los ORMs no se libran de este problema


Y las bases de datos NoSQL
tampoco.

CALLENGE IV:
BROKEN ACCESS CONTROL

A1:2021 BROKEN ACCESS CONTROL

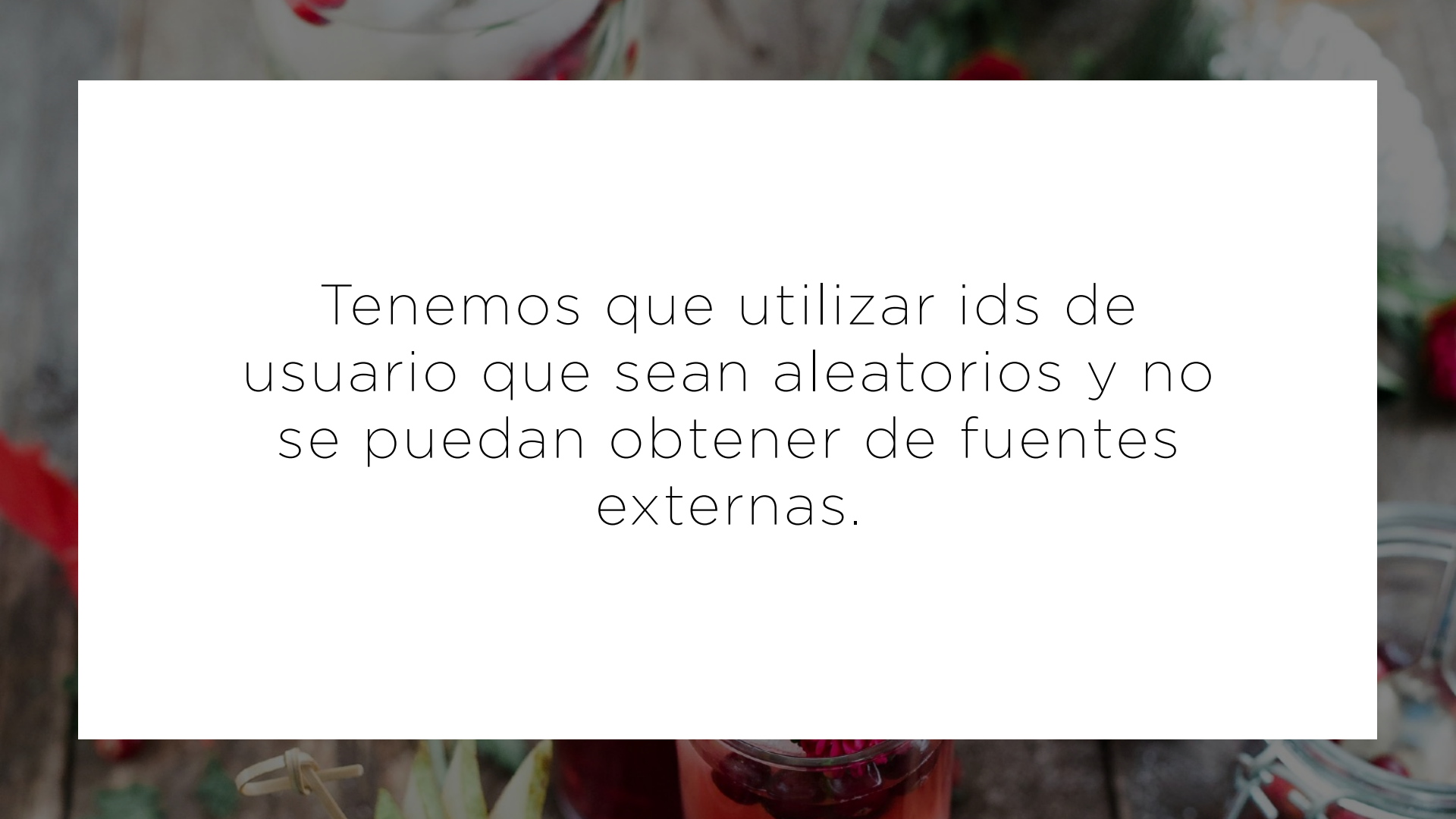
Ha subido en el ranking desde el quinto puesto que tenía anteriormente hasta el número uno.

Aquí entran múltiples variantes:

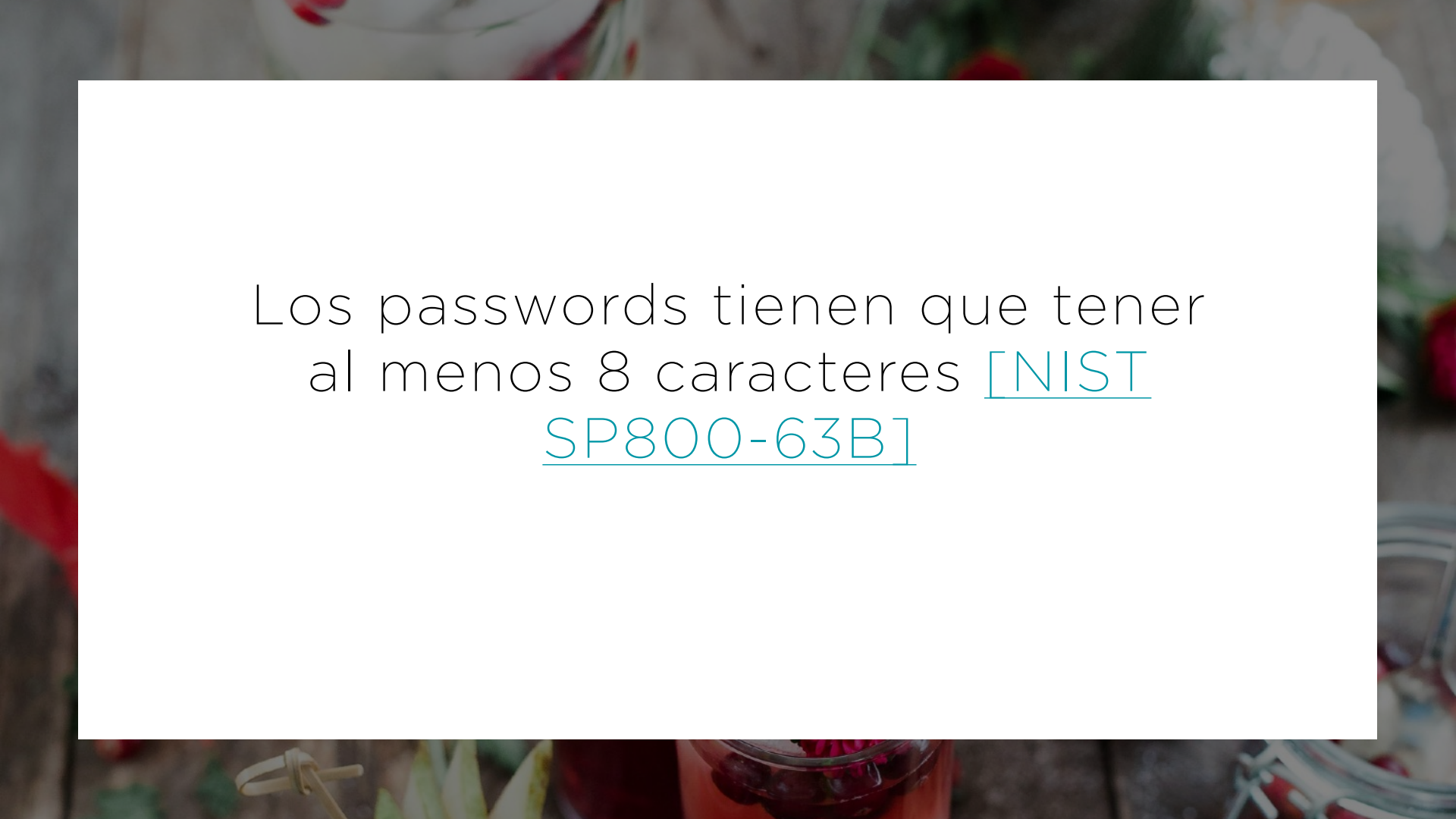
- Cuando solo usuarios con permisos deberían acceder a alguna funcionalidad pero están disponibles para cualquiera.
- Saltar controles de acceso modificando URLs o peticiones.
- Permitir ver o modificar cuentas ajenas.
- Elevación de privilegios.
- Un largo etc.


LET’S HACK!

¿CÓMO NOS PODEMOS PROTEGER?



Tenemos que utilizar ids de usuario que sean aleatorios y no se puedan obtener de fuentes externas.



Los passwords tienen que tener al menos 8 caracteres [NIST SP800-63B]

Implantar flujos de recuperación de contraseñas seguros.

RECUPERACIÓN DE CONTRASEÑAS

Algunos criterios:

- No revelar información sobre si el usuario existe o no.
- Utilizar otras vías de comunicación para comunicar la manera en la que se va a recuperar la contraseña (email por ejemplo).
- Utilizar URL tokens para una implementación simple y rápida. Estos tokens deberán ser generados de manera aleatoria, almacenados de forma segura, ser de un solo uso y tener una expiración corta.

aritz@arima.eu

https://www.linkedin.com/in/aritzberasarte/

https://twitter.com/Raskasso