KeyWords

* C#
* C++
* Java
* SQL

Interviews Questions:

1. Experience and Importance of the support, testing and develop procedures.
2. How to solve the problems of replication between two Databases.
   * Pooling or Observer
3. What is a BTree.

. Arbol balanceado:  a **B-tree** is a self-balancing tree data structure that keeps data sorted and allows searches, sequential access, insertions, and deletions in logarithmic time. The **B-tree** is a generalization of a binary search tree in that a node can have more than two children.

Uses: Index in data bases.

1. Stateless and statefull protocol, what type is HTTP?
   1. Stateless: Sin estado. Asincrono. HTTP.
   2. Statefull: Con estado. Syncrono. FTP.
2. What is AJAX and uses?
   1. **JavaScript Asíncrono y XML (AJAX)** no es una tecnología por sí misma, es un término que describe un nuevo modo de utilizar conjuntamente varias tecnologías existentes. Esto incluye: [HTML](https://developer.mozilla.org/es/docs/Web/Guide/es/HTML) o [XHTML](https://developer.mozilla.org/es/docs/Web/Guide/es/XHTML), [CSS](https://developer.mozilla.org/es/docs/Web/Guide/es/CSS), [JavaScript](https://developer.mozilla.org/es/docs/Web/Guide/es/JavaScript), [DOM](https://developer.mozilla.org/es/docs/Web/Guide/es/DOM), [XML](https://developer.mozilla.org/es/docs/Web/Guide/es/XML), [XSLT](https://developer.mozilla.org/es/docs/Web/Guide/es/XSLT), y el objeto [XMLHttpRequest](https://developer.mozilla.org/es/docs/Web/Guide/es/XMLHttpRequest). Cuando estas tecnologías se combinan en un modelo AJAX, es posible lograr aplicaciones web capaces de actualizarse continuamente sin tener que volver a cargar la página completa. Esto crea aplicaciones más rápidas y con mejor respuesta a las acciones del usuario.
3. Desgin Pattern: Singleton, Observe.
   1. Singleton:

A class of which only a single instance can exist and it’s globaly accessible. Methods: static Singleton getInstance(), the constructor is private.

* 1. Observe:

A way of notifying change to a number of classes. Communicate the changes in one class to the reset connected. 2 Interfaces: ISubject and IObserver. Methods: ISubject.AttachObserber(), ISubject.DettachObserber() and ISubject.SendNotification().

* 1. Factory- Abstract Factory:

The abstract class or interface constructor must be protected. Used in the List of the User Interface to group the same family. Abstract-Factory = Factory of factories (grouping different families). getType().

1. Create a small numerical function.

### [Senior Software Engineer at Amazon was asked...](https://www.glassdoor.ca/Interview/Amazon-Senior-Software-Engineer-Interview-Questions-EI_IE6036.0,6_KO7,31.htm)

18 Oct, 2011

|  |
| --- |
| Rand(7) from Rand(5)  [7 Answers](https://www.glassdoor.ca/Interview/Rand-7-from-Rand-5-QTN_200619.htm)  The simple solution is to implement RandBin() using Rand5(). int RandBin() { }  Sorry this UI posts without warning int RandBin() { int rand5Res = Rand5(); return rand5Res &lt; 2 ? 0 : rand5Res &lt; 4 ? 1 : RandBin(); } And then use RandBin() to implement Rand7; int Rand7() { int rand = RandBin() &lt;&lt; 2 | RandBin() &lt;&lt; 1 | RandBin(); return rand &lt; 7 ? rand : Rand7(); }  int rand7() { while(1) { int n = ((rand5()%2)\*4 + (rand5()%2)\*2 + (rand5()%2)\*1); if(n == 0) continue; return n; } } The rand5()%2 will generate 0 and 1 with equal probability and we need 3 bits since we are going from 000 upto 111. So we call this function thrice for each bit position. |

C++

int rand7(){

int x=8;

while(x>7)

x=rand5()+5\*rand5()-5;

return x;}

Otan Bruselas:

* **Aspectos de seguridad de una Aplicación web.**

En diciembre de 2001 nace el proyecto OWASP (Open Web Application Security Project o Proyecto abierto de seguridad de aplicaciones web) con el objetivo de mejorar la seguridad del software, buscar y combatir las causas que hacen un software inseguro. Este proyecto se pensó como un proyecto “abierto” de forma que cualquier profesional de la seguridad pudiera aportar su conocimiento para la causa y obtener de esta forma una metodología actualizada y bajo el punto de vista de toda la comunidad.

Su buque estrella y principal proyecto fue el **[OWASP Guide Project](https://www.owasp.org/index.php/OWASP_Guide_Project)**, una guía para el analista de seguridad donde se establecen los principales controles a auditar en cualquier aplicación web. Estos controles se pueden agrupar entre las siguientes categorías:

* **Recopilación de información**: Pruebas para obtener la mayor cantidad de información posible de la aplicación/web que sirvan posteriormente para obtener información sensible o realizar ataques que vulneren la seguridad del sitio.
* **Configuración y despliegue**: Controles de seguridad en la configuración del servidor de la aplicación, la red, conexiones a la web, etc.
* **Gestión de la Identidad**: Controles de seguridad en la definición de roles, usuarios, numeración de cuentas y políticas de acceso en la aplicación/web.
* **Autenticación**: Controles de seguridad relacionado con el acceso a la aplicación/web como las contraseñas.
* **Autorización**: Controles de seguridad para probar las posibles escaladas de privilegios o evasión de métodos de autorización en el portal.
* **Control de la sesión**: Controles de seguridad relacionados con las cookies y tiempos de sesión.
* **Validación de entrada de datos**: Controles de seguridad en todas las entradas de información que disponga la aplicación/web (formularios de contacto, búsquedas, registros de información).
* **Manejo de errores**: Seguridad y correcto funcionamiento de los mensajes de error que muestra la aplicación.
* **Criptografía**: Controles para probar la seguridad y robustez de los mecanismos de cifrado que implemente la aplicación/web.
* **Lógica de la aplicación**: Controles de comportamiento de la aplicación frente a usos inesperados que pueden ocasionar brechas de seguridad.
* **Pruebas de cliente**: Todas las pruebas de seguridad realizadas desde el punto de vista del usuario.

La guía cuenta actualmente con un total de 87 controles diferentes que engloban pruebas de seguridad relacionadas con estas categorías y que permiten obtener un estado global y actual de la seguridad de la página web o aplicación.

* **Aspectos de seguridad en general: Cyber Seguridad**

## 1. Follow the OWASP Top Ten

I’ve already covered this in greater depth, [in a recent post](https://blog.sqreen.com/owasp-top-10-cheat-sheet-startup-ctos/). But, it’s still a crucial list to keep in mind. If you’re not familiar with the OWASP Top Ten, it contains the most critical web application security vulnerabilities, as identified and agreed upon by security experts from around the world.

These security vulnerabilities target the confidentiality, integrity, and availability of an application, its developers, and its users. They cover such attack vectors as injection attacks, authentication and session management, security misconfiguration, and sensitive data exposure.

By being aware of them, how they work, and coding in a secure way the applications that we build stand a far better chance of not being breached. Doing so also helps you avoid being on any [end of year hack list](https://blog.sqreen.com/security-year-in-review/).

The list, surprisingly, doesn’t change all that often. Sadly, many of the same issues seem to remain year after year, despite an ever growing security awareness within the developer community.

This is both a blessing and a curse. As they don’t change often, you can continue to review the preparedness of your application in dealing with them. [The latest list was published in 2017](https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf).

## 2. Get an Application Security Audit

Let’s assume that you take the OWASP Top Ten seriously and your developers have a security mindset. Let’s also assume that they self-test regularly to ensure that your applications are not vulnerable to any of the listed breaches. You may even have a security evangelist on staff.

While these are all excellent, foundational steps, often they’re not enough. This is because of preconceived biases and filters. Your team lives and breathes the code which they maintain each and every day. Because of that, over time, they’ll not be able to critique it objectively. Increasingly, your team will be subjective in their analysis of it.

It’s for this reason that it’s important to get an independent set of eyes on the applications. By doing so, they can be reviewed by people who’ve never seen them before, by people who won’t make any assumptions about why the code does what it does, or be biased by anything or anyone within your organization either.

Additionally, they will be people with specific, professional application security experience, who know what to look for, including the obvious and the subtle, as well as the hidden things. They’ll also be abreast of current security issues and be knowledgeable about issues which aren’t common knowledge yet.

Specifically, what I’m suggesting is to get an application security audit carried out on your application. They can give you a baseline from which to grow.

This can be potentially daunting if you’re a young organization, one recently embarking on a security-first approach. But, setting concerns aside, security audits can help you build secure applications quicker than you otherwise might.

## 3. Implement Proper Logging

Now that you’ve gotten a security audit done, you have a security baseline for your application and have refactored your code, based on the findings of the security audit, let’s step back from the application.

Let’s now look at the bigger picture, and look at the outside factors which influence the security of an application. Specifically, let’s look at logging.

Invariably something will go wrong at some stage. There’ll be a bug that no one saw (or considered severe enough to warrant particular attention) — one that will eventually be exploited.

When that happens, to be able to respond as quickly as possible — before the situation gets out of hand — you need to have proper logging implemented.

Doing so provides you with information about what occurred, what lead to the situation in the first place, and what else was going on at the time. As the saying goes: proper preparation prevents poor performance.

To do so, first, ensure that you’ve sufficiently instrumented your application. Depending on your software language(s), there is a range of tools and services available, including [Tideways](https://tideways.com/), [Blackfire](https://blackfire.io/), and [New Relic](https://newrelic.com/).

Secondly, store the information so that it can be parsed rapidly and efficiently when the time comes. There is a range of ways to do this. From simple solutions such as [the Linux syslog](http://man7.org/linux/man-pages/man3/syslog.3.html), to open source solutions such as [the ELK stack](http://logz.io/learn/complete-guide-elk-stack/#What%20is%20the%20ELK%20Stack) (Elasticsearch, Logstash, and Kibana), to SaaS services such as [Loggly](https://www.loggly.com/), [Splunk](https://www.splunk.com/), and [PaperTrail](https://papertrailapp.com/).

Regardless of what you use, make sure that the information is being stored and that it’s able to be parsed quickly and efficiently when the time comes to use it.

## 4. Use Real-time Security Monitoring and Protection

Any consideration of application security would be incomplete without taking classic firewalls and web application firewalls (WAFs) into consideration.

[As I wrote about recently](https://blog.sqreen.com/why-web-application-firewalls-fail-to-protect-web-applications/), firewalls, while effective at specific types of application protection, aren’t the be all and end all of application security.

WAFs fall short for a number of reasons, including that they can generate a large number of false positives and negatives, and can be costly to maintain. However, they do afford some level of protection to your application.

So, if you want to use a WAF, I suggest that you either use them in addition to a Runtime Application Self-Protection (RASP) tool, or use Application Security Management platforms [such as Sqreen](https://www.sqreen.com/) that can provide RASP and in-app WAF modules tuned to your needs, to provide real-time security monitoring and protection.

That way, you can protect your application from a range of perspectives, both internal and external.

## 5. Encrypt Everything

Now that your application’s been instrumented and has a firewall solution to help protect it, let’s talk about encryption. And when I say encryption, I don’t just mean using HTTPS and HSTS. I’m talking about encrypting all the things.

I believe it’s important to always use encryption holistically to protect an application. This might seem a little Orwellian, but it’s important to consider encryption from every angle, not just the obvious or the status quo.

It’s great that services such as [Let’s Encrypt](https://letsencrypt.org/) are making HTTPS much more accessible than it ever was before. And it’s excellent that such influential companies as Google are rewarding websites for using HTTPS, but this type of encryption isn’t enough.

It’s important to also make sure that data at rest is encrypted as well. HTTPS makes it next to impossible for Man In The Middle (MITM) attacks to occur.

But if someone can get to your server (such as a belligerent ex-staffer, dubious systems administrator, or a government operative) and either clone or remove the drives, then all the other security is moot.

So, please don’t look at security in isolation, or one part of it. Look at it holistically and consider data at rest, as well as data in transit.

## 6. Harden Everything

Now that all traffic and data is encrypted, what about hardening everything? From operating systems to software development frameworks you need to ensure that they’re sufficiently hardened.

This is too complex a topic to cover in the amount of space I have available in this article. So let’s instead consider a concise list of suggestions for both operating systems and frameworks.

* Is your web server using modules or extensions that your application doesn’t need?
* Is your software language using modules or extensions that it doesn’t need?
* Does your software language allow remote code execution, such as exec and proc to occur?
* What’s the maximum script execution time set to?
* What access does your software language have to the filesystem?
* Where is session information being stored?
* Are the servers, services (such as MySQL, PostgreSQL, and Redis) and software language configuration files write protected?
* Are your servers using security extensions such as [SELinux](https://selinuxproject.org/page/Main_Page) or [AppArmor](https://wiki.ubuntu.com/AppArmor)?
* Is incoming and outgoing traffic restricted?
* What users are allowed to access the server and how is that access managed?

How do your servers, services, and software language configurations fare? This is a complex topic. So, here is a short list of best practice guides to refer to:

* [Ruby on Rails Security Guide](http://guides.rubyonrails.org/security.html)
* [PHP Security Checklist](https://www.sqreen.com/resources/php-security-checklist?utm_source=blog)
* [Ruby Security Handbook](https://www.sqreen.com/resources/ruby-security-handbook?utm_source=blog)
* [Python Security](https://blog.sqreen.com/preventing-sql-injections-in-python/)
* [Node.js Security Handbook](https://www.sqreen.com/resources/nodejs-security-handbook?utm_source=blog)
* [Hardening the Linux server](https://www.ibm.com/developerworks/linux/tutorials/l-harden-server/)

## 7. Keep Your Servers Up to Date

In addition to ensuring that your operating system is hardened, is it up to date? It could very well be hardened against the current version, but if the packages are out of date (and as a result contain vulnerabilities), then there’s still a problem.

Make sure that your servers are set to update to the latest security releases as they become available. I’m not suggesting updating each and every package, but at least the security-specific ones.

Depending on your organization’s perspective, you can elect to automate this process. Alternatively, you can review and approve updates individually.

If you want to automatically install security upgrades, you can use:

* [UnattendedUpgrades](https://wiki.debian.org/UnattendedUpgrades) on Debian-related distributions
* [yum-cron](https://access.redhat.com/discussions/1238193) with “update\_cmd = minimal-security-severity:Important”
* [The Automatic Updates feature in Windows](https://support.microsoft.com/en-us/help/306525/how-to-configure-and-use-automatic-updates-in-windows)

If you’re not using one of these, please refer to the documentation for your operating system or distribution.

## 8. Keep Your Software Up to Date

As well as keeping the operating system up to date, you need to keep your application framework and third party libraries up to date as well.

Some people may scoff at the thought of using a framework. That’s not a debate that I’m going to engage in today, suffice to say that they both have their place, and when used well, can save inordinate amounts of time and effort.

Frameworks and third-party software libraries, just like operating systems, have vulnerabilities. If they’re properly supported, then they will also be rapidly patched and improved. Given that, it’s important to ensure that you’re using the latest stable version — if at all possible.

Most languages, whether dynamic ones such as PHP, Python, and Ruby, or static ones such as Go, have package managers. These tools make the process of managing and maintaining external dependencies relatively painless, as well as being automated during deployment.

Ensure that you take advantage of them and stay with as recent a release as is possible.

## 9. Stay Abreast of the Latest Vulnerabilities

This is strongly tied to the previous point. Given the number of attack vectors in play today, vectors such as Cross-site scripting, code injection, SQL injection, insecure direct object references, and cross-site request forgery it’s hard to both stay abreast of them as well as to know what the new ones are.

But, such is life. Given the world in which we live and the times in which we operate, if we want to build secure applications we need to know this information. Gladly, there are a range of ways in which we can get this information in a distilled, readily consumable fashion.

Sqreen does a bi-weekly newsletter roundup of interesting security articles you can subscribe to. Here is a list of blogs and podcasts you can regularly refer to, to stay up to date as well:

### Blogs

* [Troy Hunt](https://www.troyhunt.com/): The Australian Microsoft Regional Director and MVP. He also tweets at [@troyhunt](https://twitter.com@troyhunt/).
* [Krebs on Security](http://krebsonsecurity.com/) by Brian Krebs. Brian is an independent investigative journalist, specializing in cybercrime. He also tweets at [@briankrebs](https://twitter.com@briankrebs/).
* [Dark Reading](http://www.darkreading.com/): one of the most widely read cybersecurity news sites. It reports on attacks and the key ways to defend yourself against them.
* The Guardian’s [Data and Computer Security section](https://www.theguardian.com/technology/data-computer-security). An excellent source of the latest information on what’s happening around the world with respect to security.
* [Schneier on Security](https://www.schneier.comr/) by Bruce Schneier. Bruce’s been writing about security since 2004 and is the Chief Technology Officer of Resilient and a board member of [the EFF (Electronic Frontier Foundation)](https://www.eff.org/).

### Podcasts

* [OWASP Podcast](https://www.owasp.org/index.php/OWASP_Podcast)
* [Crypto-Gram Security Podcast](http://crypto-gram.libsyn.com/)
* [Risky Business](http://risky.biz/)
* [Down the Security Rabbithole](http://podcast.wh1t3rabbit.net/)
* [Defensive Security](https://www.defensivesecurity.org/category/podcast/)

## 10. Never Stop Learning

Finally, perhaps this is a cliché, but never stop learning. You may be all over the current threats facing our industry. But that doesn’t mean that new threats aren’t either coming or being discovered.

Given that, make sure that you use the links in this article to keep you and your team up to date on what’s out there. Then, continue to engender a culture of security-first application development within your organization.

That way, you’ll always have it as a key consideration, and be far less likely to fall victim to security or data breaches.

## In Conclusion

That’s been 10 best practices for securing your web applications. No one article is ever going to be able to cover ever topic, nor any one in sufficient depth. The security landscape is changing far too quickly for that to be practical.

However, with the information here, you’re equipped with 10 best practices to guide you on your journey to building secure applications. Make sure that you use them and consider security as equally as important as testing and performance.

* Cual es tu debilidad?
* Cuales son tus puntos fuertes?
* Porque eres el mejor para este puesto?
* Ejercicio sobre ListBox.
* Ejercicio sobre parsing ficheros.