Security analysis, strategy, pentesting

The most delicate data: financial transactions, medical information, and in general very personal information regularly passes through our digital infrastructure.

A very important question that arises is: "how do we keep it secure?".

Security is not an afterthought, and it's also not a project that is done at some point. Security is an ongoing *process* that requires constant and regular attention. Every time we create some new functionality in a web application, we should always consider the risk that some new endpoint or business logic exposes us to security breaches.

Guaranteeing security in a web application requires a mixed approach. Here are the main three categories of security concerns that I like to focus on.

The basics/OWASP top 10

Making sure that the foundation of our security is in order means having up to date components, having solid cryptography in place, avoiding injection attacks (an attacker puts some code in a form input and the backend accidentally ends up executing it), and having a solid configuration, among other things.

There is nothing banal about these basics, but they are relatively standard and almost straightforward to test and improve. It's "just" a matter of organizing a pentest, running some standardized scripts, and apply the improvements depending on severity.

The performance/DDoS

A nastier kind of vulnerability is related to Denial of Service attacks, where an attacker exploits a resource-intensive feature of the application to bring infrastructure down.

For example, a search endpoint might be uncached (because every user searches something else) and an attacker might automatically send hundreds of search queries per second.

The effect of this might either be that the infrastructure collapses, rendering the web application unable to perform its function, or even worse causing automatic scaling of cloud infrastructure resulting in sky-high bills from the cloud provider.

In both cases, we need security protection in the form of some protected CDN such as Cloudflare, rate-limiting, captchas, and also plain old performance improvements which will not only improve the safety of the platform: they will also improve the end user experience.

The semantics/access and authorisation

The last kind of vulnerability is the one ignored the most during pentests, but it is also the most dangerous. Every API endpoint that exposes user data needs to be tested:

- when we are logged in as an entitled user, do we get exactly and only the data that we are allowed to see?
- when we are logged in as a non-entitled user, does the right data get filtered and not exposed by the backend?

- when we are not logged in, does the data also get filtered correctly by the backend?

Also, there are corollary questions. For example, does the system perform more processing for some anonymous inputs? Sometimes an attacker will be able to guess the existance or not of a given username just by measuring how long a failed login attempt takes: if it takes longer, it means that the username was found, but the password was not correct.

All of these attacks are *semantic*-level attacks, meaning that one needs to carefully look at the application and its endpoints one by one and think about the underlying business logic, and a one-size-fits-all solution/script cannot be unleashed once and that's it.

In short

Ensuring security of a web application needs a mix of standardized checks, performance hardening, and plain-old analysis of the business requirements/business rules/business logic of the application to make sure that the right data is filtered correctly.