**Description**

Exposed Insecure Functionalities are vulnerabilities that typically emerge in infrastructures or applications due to poorly implemented (or non-existent) security controls which, in turn, expose potentially critical or sensitive functions to the open internet. Exposed Insecure Functionalities are one class of origin for information exposure resting under the broader OWASP Top 10 Security Misconfigurations classification.

Often during the development phase of a server or web application build, code is added by the developer for ease of access when testing and debugging. As is so often the case though, what was originally intended as a benign aid for increased efficacy and quality can dually serve as an entry point for malicious actors simply because the security risk was not considered at the beginning. Thus, this insecure *back door* code can make its way into production, suggesting that internal security procedures and processes are not in place or enforced to ensure adequate application and system hardening prior to deployment.

Exposed Insecure Functionalities are particularly useful to attackers performing reconnaissance activities as they will often leak application and system configuration and deployment details to remote users.

**Impact**

There are countless real-world examples of Exposed Insecure Functionalities introducing security vulnerabilities into previously secure environments, with the level of impact ranging from the release of sensitive information, through to the complete control over the web application and server, as well as affording a stepping stone into additional access systems.

For example, infamous breaches such as the leak of 154 million voter records (including addresses, phone numbers, marital status, estimated incomes, and political parties) [hit](https://www.helpnetsecurity.com/2016/06/23/154-million-us-voter-records-exposed/) [more than one country](https://www.databreaches.net/personal-info-of-93-4-million-mexicans-exposed-on-amazon/) due to exposed and misconfigured databases.

**Scenarios**

The following example supposes a web application that exposes a vulnerable authentication mechanism. The application interprets an additional, optional parameter named debug - plausibly a leftover of the development phase - as a request to switch to debug mode, and that the username and password go unchecked when this parameter is issued.

If the debug parameter is a known issue, it is a straightforward matter to bypass the authentication process. If it's not, it might be discovered by motivated attackers by running automated scanners designed to find suspicious hidden parameters.

This means that a request to log in as administrator, as in the following, would be rejected because of the incorrect password:

POST /auth

user=admin&pass=wrong

While a malicious actor could add the debug flag to the request to obtain unauthorized access to the application.

POST /auth

user=admin&pass=wrong&debug=1

**Prevention**

Developers must remove all debugging and test functionality from production applications and systems if a justifiable business need does not exist. Application build procedures must include steps to remove all files and features which are unnecessary for a production deployment, and these steps must be adhered to by the project, and development team.

Documented internal security processes and controls should confirm this has occurred prior to production release; debugging features should never be shipped into production environments.

Developers must adhere to this hardening process both before and after an application has moved into a production environment. Application directories should be subject to regular periodic reviews to ensure that unnecessary or legacy components do not re-appear.

**Testing**

Verify that web or application server and application framework debug modes are disabled in production to eliminate debug features, developer consoles, and unintended security disclosures.

* **OWASP ASVS**: [14.3.2](https://github.com/OWASP/ASVS/releases/download/v4.0.2_release/OWASP.Application.Security.Verification.Standard.4.0.2-en.pdf)
* **OWASP Testing Guide**: [Test Application Platform Configuration](https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/02-Configuration_and_Deployment_Management_Testing/02-Test_Application_Platform_Configuration)