Ten Tips For Secure Magento 2 Development

EXT DN

Magento Extension Developers Network

VALIDATE AND SANITISE ALL INPUTS

Any input your code receives from anywhere is potentially bad. Don’t trust anything, even if it comes from a trusted person like an admin, because accounts get compromised. Either sanitise inputs so they contain only good data (removing invalid characters, casting numeric values to int or float, etc.), or return an error message if you find anything bad or unexpected. Inputs must be validated or sanitised on the server; JavaScript validation can be bypassed.

AUTHENTICATE USERS AND CONTROL ACCESS CAREFULLY

All admin and API features should have a specific Access Control List resource to allow or deny access to that feature or data. In general it is a good idea for a feature to only require the minimum access in its ACL to execute successfully. All customer features should verify that data belongs to only the current customer to prevent data leakage and unauthorised access. Magento does provide a `CustomerSession` object to manage customer sessions, rely on that first.

AVOID DYNAMIC CODE TO PREVENT REMOTE CODE EXECUTION (RCE)

PHP offers numerous ways to execute code dynamically, including `eval` , `exec` , `system` , `shell_exec` , `create_function` , `preg_replace` with /e, and more. Not using these methods at all is the best way to avoid remote code execution vulnerabilities.

BE CAREFUL ABOUT DATABASE QUERIES TO PREVENT SQL INJECTION (SQLI)

Use Magento’s database abstraction layer rather than constructing SQL queries manually. It will protect you under most circumstances. Any user input passed to the database should be checked to make sure it doesn’t contain unexpected input.

AVOID UNSERIALIZE() TO PREVENT PHP OBJECT INJECTION (POI)

If you are in a position to decide data storage formats, JSON will almost always be an acceptable and safer option than PHP's serialization. When working with already-serialised data, use Magento's provided classes like `\Magento\Framework\Unserialize\Unserialize` and `\Magento\Framework\Serialize\SerializerInterface` .

ESCAPE OUTPUT TO PREVENT CROSS-SITE SCRIPTING (XSS)

Any time you output a variable, use the appropriate Magento escape method to avoid running injected code or breaking the page. These include `$block->escapeHtml($string)` , `escapeHtmlAttr` , `escapeJs` , `escapeCss` , and `escapeUrl` (in Magento 2.2+).

USE FORM KEY VALIDATION TO PREVENT CROSS-SITE REQUEST FORGERY (CSRF)

Any request that changes data or causes an action to occur should be sent as a POST request, and include a form key that is specific to the user's session. That form key must be validated on the server side, and an error should be returned with no action taken if it fails to validate. Form keys should never be passed in URLs or over a non-SSL connection.

VALIDATE UNKNOWN XML TO PREVENT EXTERNAL ENTITY PROCESSING (XXE)

Any XML parsed by `simplexml_load_string()` is potentially vulnerable to XML External Entity Processing. Check the XML with `\Magento\Framework\Xml\Security->scan($xml)` before parsing to stay safe.

AVOID LOGGING CONFIDENTIAL DATA TO PREVENT INFORMATION DISCLOSURE

Log files should never contain passwords, API Keys, credit card numbers, or any other confidential data, even in error messages or stack traces. Avoid logging any form of customer-specific information. Log files are often shared, and easily misplaced or forgotten.

MONITOR YOUR DEPENDENCIES

When you include third-party code as a dependency, you are taking on responsibility for it. Follow their releases and security updates to ensure you are in a position to make any necessary changes to your code quickly. Use tools like **Github Security Alerts** and **Roave/SecurityAdvisories** to keep up to date.