Context	Types of Plug-and-Play Vulnerabilities	Consequences	Mitigation
Plug-ins Installation	Incorrect user notification of plug-in permissions	Gain privileges, Spoofing, User-assisted attack	- *Central install point*: all the installation requests are guaranteed to go through this component, who is in charge of (i) consistently generating install warning prompts before any install requests; (ii) showing all the requested permissions
	Bypassing user notification for plug-in installation	Stealth installation of malicious plug-ins	
	Lack of plug-in's configuration file sanitization	Arbitrary code execution, Privilege elevation, Data leakage, Directory path traversal, Application crash	- *Configuration validator*: it ensures that configuration files are not used as an attack vector, through typed parsing and validation of configuration files.
	Improperly checking the origin of an install request	Spoofed origin of an install request, Misleading the user to install a malicious plug-in	- *Whitelist of install origins*: Specification of the only allowed install points that can trigger the installation, avoiding that malicious code or other vectors silently install a plug-in.
Update Plug-ins	Elevation of privilege through a plug-in update	Privilege elevation	- *Lifetime enforcement of plug-in permissions*: Comparing against previously list of permissions provided by plug-in during install
Plug-in Registry Management	Extraction/storage of plug-in with world-readable/writable permissions or in unsafe directories	Modify/Erase plug-in data, Replace a benign plug-in with a malicious one, Execute unauthorized code, Symlink attack	*Dedicated secure storage*: extract the software bundle to the application's dedicated folder, with restricted access.
Plug-and-Play Execution Environment	Lack of compartmentalization of plug-ins	Arbitrary code execution	- *Compartmentalization of plug-ins*: Each plug-in must be encapsulated in a separate compartment. - *Isolated object domains*: Each compartment must have its own copy of objects for communication with the PnP environment.
	Lack of fine-grained and modular permission setting	Overprivileged plug-in	- Fine-grained, modular, permission assignment. - Declarative-based request for accessing data/functionality.
	Allowing a plug-in to elevate its permission by manipulating (or delegating a task to) a process in the PnP environment that has higher privileges	Arbitrary code execution, Privilege elevation	- *Limit plug-ins exposure to OS processes*: Leveraging a mechanism that intermediates any system call between the sandboxed child process and the underlying OS to prevent the low-privileged process to attempt to communicate with other higher-privileged processes. - *Limit plug-ins exposure to High privilege PnP APIs*: The access that plug-ins have to higher privileged APIs provided by the core application should be limited according to the permissions they have asked for and the privileges they have.
	Improper object access control in compartmentalized PnP environment	Arbitrary code execution, Disrupt the PnP execution environment, Privilege elevation	- *Security Policy Enforcement through Object Wrappers:* Adoption of different types of object wrappers that acts as proxies for a real object residing in a different compartment. These wrappers apply a security policy which enforces what type of properties and operations would get accessed by the callee compartment depending on the relationship between the caller and the callee compartments.
	Unsanitized plugin data	Cross-site scripting (XSS), Steal credentials, Code injection, Arbitrary code execution, Memory corruption, SQL Injection	- *Input validation of incoming plug-in data*: Data transferred by plug-ins to the PnP Environment must be sanitized.
	Improper origin check of requests by plug-ins	Plug-ins tampering with other plug-ins, Data leakage to unintended plug-ins, Arbitrary code execution, Same-origin policy bypass	- *Origin check*: Verifying request origins and authenticating plug-ins requests against a security policy.
	Improper isolation of objects used by plug-ins in PnP environment	Override intended extension behavior, Data leakage to unintended plug-in, Bypass protection mechanism, Application crash, Sandbox and compartment escape, Overwrite memory, Code injection	- *Isolated object domains*: Each plug-in has its own copies of objects inside PnP Environment, minimizing the risk of the same object being used by another plug-in. The PnP environment manages these objects (which object is owned by who) and enforces that these objects are not used as an attack vector.
Plug-ins Request Handling	Plug-ins requests are handled without authorizing plugins that initiate the request	Plug-ins tampering with other plug-ins, Data leakage to unintended plug-ins, Arbitrary code execution	- *Authorize the source of request*: Upon a request, PnP host must authorize the plugin that initiates a request or subscribes to an event. - *Decomposition of events:* events are decomposed into sensitive and non-sensitive events. Listeners can only subscribe to sensitive events if and only if they have enough permissions. - *Hide PnP internal events:* Events and APIs specific to PnP environment must be hidden from plugins.
	Reentrant event callbacks	Unexpected state, DoS: PnP environment crash	- *Atomic event dispatcher*: dispatch events in an atomic fashion