

Threat Modeling the Game Mod Ecosystem

A structured threat model for game modifications, loaders, community distribution, and the boundaries between data and executable code.

Publication: October 2026 | Technical analysis, not an experimental benchmark | Read online: research.malguard.site/papers/game-mod-threat-model.html

Why mod security needs its own model

Game modifications are not a single file type. They may be textures, saved configurations, scripts, native plugins or bundled installers. Their capabilities differ dramatically. A texture replacement and a native code plugin should not receive the same risk assumptions simply because both are called mods.

Security assessment begins with the relationship between the game, mod loader, operating system and distribution channel. The goal is to identify trust boundaries and possible failure modes, not to stigmatize modding communities.

Assets and trust boundaries

Assets worth protecting include the player's account credentials, local files, game installation, browser sessions and system integrity. Trust boundaries include the download site, archive parser, installer, mod loader, game process and any services contacted by installed components.

The risk changes when a package crosses from inert game data into script execution or native code. A loader that grants broad process permissions expands the potential impact of any unsafe plugin.

Threat categories

Distribution risks include impersonated releases, misleading mirrors, altered packages and malicious updates. Package risks include unexpected executable components, hidden dependencies and resource-exhaustion hazards during parsing. Runtime risks include code running with the user's privileges and unexpected network or filesystem access.

MITRE ATT&CK documents abuse of Windows DLL loading behavior. This is relevant to native plugin ecosystems as a threat category, but the presence of a DLL is not evidence that the technique is being used.

Defensive controls by stage

Before installation, compare the release against authenticated publisher information and inspect the package structure. At installation, favor minimal privileges, clear file changes and user-visible permissions. At runtime, keep games and mod loaders updated, avoid unknown executables, and investigate unexpected behavior with appropriate defensive tools.

Controls must be matched to the stage. A checksum cannot replace provenance; an archive inventory cannot replace behavioral analysis; a runtime observation cannot retroactively establish the origin of a package.

Hypothetical case study

A fictional mod page advertises a graphics enhancement. The downloaded package contains assets and an additional installer. The extra installer is not proof of malware, but it introduces a new trust boundary and executable capability. A responsible scanner would highlight the discrepancy and explain what it has not verified.

The user should be able to decide whether the source and installation method are credible without being shown a fabricated safety guarantee.

Research limits

This paper provides a conceptual threat model, not incident attribution or a dataset of malicious mods. It does not describe payload construction, bypasses or exploitation steps. Future work could validate the model against consented, non-malicious test packages and document false alarms.

Evidence-to-claim mapping

A useful assessment should state what each observation actually supports. A digest match supports the narrow statement that two retrieved byte sequences match; it does not establish publisher identity. An archive inventory supports claims about visible entries and declared formats; it does not establish runtime behavior. An authenticated release signature may strengthen attribution to a signing key, but cannot by itself prove the signed code is benign.

For each claim, record the artifact identifier, evidence source, collection time, analysis scope, confidence and unresolved questions. Treat absence of evidence as unknown when coverage is incomplete. This prevents a report from quietly changing “not observed” into “safe.”

Threat-model review checklist

Before publishing an assessment, ask whether the package is primarily data, script, native code or an installer; which component can execute; which user privileges it receives; which distribution party is being trusted; and which capabilities were actually observed rather than inferred. If the evidence is incomplete, identify the missing checks explicitly.

This checklist is an analytical aid, not a substitute for a verified software origin, controlled testing or independent review. Its value lies in making assumptions inspectable and allowing another analyst to disagree with a specific claim rather than a vague safety label.

References and further reading

These external references support background concepts. The analysis and examples above are original to MalGuard Research; they are not attributed experiments.

- MITRE ATT&CK: Hijack Execution Flow, DLL · <https://attack.mitre.org/techniques/T1574/002/>
- NIST Secure Software Development Framework · <https://csrc.nist.gov/pubs/sp/800/218/final>