R

Replay Attack: How It Works and How to Prevent Token Reuse

A replay attack resends a previously valid message, token, or signed request. Learn how it differs from MITM, why encryption is not enough, and which anti-replay controls work.

PUBLISHER
Gridinsoft Support

UPDATED
Oct 8, 2026

READ TIME
6 min

CONTENTS

In this guide

01 What is a replay attack?

02 How a replay attack works

03 Replay attack vs. related attacks

04 Can encryption stop replay attacks?

05 How to prevent replay attacks

06 How to detect possible replay

07 What to do after suspected token or request replay

08 Frequently asked questions

09 References

01 What is a replay attack?

A replay attack happens when an attacker captures a valid message, request, credential artifact, or signed transaction and sends it again so that a system accepts the old data as new. The attacker may not need to read or modify the captured data. The weakness is that the receiver verifies authenticity but does not adequately verify *freshness* or whether the message has already been used.

Replay attacks can affect authentication exchanges, API requests, payment instructions, access tokens, wireless protocols, industrial messages, and signed blockchain transactions. The result can be an unauthorized login, a duplicated action, or use of a valid request in the wrong context.

02 How a replay attack works

1. A legitimate client sends an authenticated request, token, or signed message.
2. The attacker obtains a reusable copy through network access, malware, logs, browser storage, an exposed proxy, or another compromise.
3. The attacker submits the same artifact later, sometimes from a different device or session.
4. The server validates the signature or credential but fails to detect that the request is old, duplicated, expired, or out of context.
5. The protected action runs again or the attacker receives access.

For example, a correctly signed funds-transfer request may still be dangerous if the server accepts the same request identifier twice. The signature proves that the message is authentic and unchanged; it does not automatically prove that this is the first time the server has seen it.

03 Replay attack vs. related attacks

Attack	What the attacker does	Key distinction

Replay attack	Reuses a previously valid message or artifact	Exploits missing freshness or duplicate detection
Man-in-the-middle	Positions between parties to observe or alter communication	Can be a way to capture data, but replay does not always require an active MITM
Session hijacking	Uses or takes over a valid session	Often involves a stolen session secret; replay is one possible method
Credential stuffing	Tries username-password pairs stolen from another service	Reuses credentials across accounts or sites rather than replaying a protocol message
CSRF	Tricks a signed-in browser into issuing an unwanted request	The victim's browser sends the request; the attacker may never possess the session token

04 Can encryption stop replay attacks?

Encryption is necessary but is not a complete application-layer replay defense. Modern protected transports can reject duplicated records inside an established connection. However, an application can still accept a stolen bearer token, cookie, signed URL, or valid business request submitted through a new protected connection if that artifact remains valid and reusable.

Message authentication and digital signatures protect integrity and authenticity. Replay resistance additionally requires a changing value or server state that lets the receiver distinguish a new request from a captured one.

05 How to prevent replay attacks

Use a unique nonce or request identifier

Generate a value that is unique within the relevant security context and include it in the authenticated or signed data. The receiver must record or otherwise validate it and reject reuse. Merely adding a random field without checking it does not prevent replay.

Enforce timestamps and narrow acceptance windows

Sign the timestamp and reject requests outside a short, documented window. Account for clock skew and protect time synchronization. A timestamp limits the opportunity but still permits replay inside the accepted window unless combined with a nonce, sequence, or duplicate check.

Use sequence numbers or monotonic counters

Reject a message with a sequence number that has already been accepted or falls outside an approved anti-replay window. Define recovery behavior for retries, reordering, device resets, and concurrent clients.

Bind proof to the intended context

Include the method, destination, resource, amount, audience, account, and protocol or chain identifier in what is authenticated. Context binding helps stop a valid signature from being reused for a different endpoint, tenant, transaction, or network.

Limit and protect tokens

- Use short-lived access tokens and rotate refresh tokens.
- Invalidate sessions after compromise, password reset, or privilege change.
- Prefer sender-constrained or proof-of-possession tokens where the risk justifies them.
- Do not expose secrets in URLs, analytics, application logs, referrers, or client-side code.
- Use secure, HttpOnly, and appropriate SameSite cookie settings for browser sessions.

Make state-changing operations idempotent

For payments, orders, and provisioning APIs, accept an idempotency key and store the result. A retry with the same key should return the original result rather than execute the operation again. Set retention long enough to cover realistic retry and attack windows.

06 How to detect possible replay

- The same token ID, nonce, signature, request body hash, or idempotency key appears more than once.

- A previously used artifact arrives from a new device, geography, client certificate, or network.
- Identical state-changing requests occur in an implausibly short period.
- Sequence numbers move backward, repeat, or fall outside the accepted window.
- Many requests fail timestamp, nonce, or token-reuse validation.

Keep authentication and transaction logs with synchronized timestamps, but avoid storing reusable secrets. Alerting should distinguish normal client retries from a duplicate that could cause a second side effect.

07 What to do after suspected token or request replay

1. Revoke the affected session, token family, API key, or certificate as appropriate.
2. Stop duplicate processing and determine which actions actually completed.
3. Preserve request IDs, timestamps, source details, relevant logs, and audit records.
4. Find how the artifact was exposed: endpoint malware, XSS, logs, proxy access, insecure storage, or a protocol flaw.
5. Notify affected users or business owners and reverse unauthorized transactions when possible.
6. Add freshness and duplicate controls, then test normal retries, failures, concurrency, and clock drift.

08 Frequently asked questions

Does multifactor authentication prevent replay?

Not automatically. Some authentication protocols are designed to resist replay, but a stolen post-login session token may still be reusable. Protect both authentication and session management.

Is a one-time password replay-proof?

Only when the verifier rejects reuse and enforces the correct time or counter window. A phished code may still be relayed and used before it expires.

Are nonces secret?

Usually they do not need to be secret. They need to be unique in the required context and covered by the message authentication or signature so an attacker cannot substitute one.

09 References

- NIST CSRC: nonce definitions and replay protection
- NIST SP 800-63B: threats and security considerations
- MDN: Replay attack

NEED MORE HELP?

We are here when the guide is not enough.

Visit the Gridinsoft Support Center for current product guidance or submit a support request with the details of your case.

Open this guide online at support.gridinsoft.com