What is prompt obfuscation?

Prompt obfuscation is encoding a prompt (leetspeak, Base64, look-alike Unicode, zero-width characters, simple ciphers) so a safety filter reads it as harmless while the language model still decodes and follows it. It is a delivery method for jailbreaks and prompt injection, not a separate flaw. Tools such as Parseltongue bundle hundreds of these transforms so an attacker can try many encodings fast. The defence is to canonicalise and decode input before you filter it, and to judge intent on the decoded text.

HOW IT WORKS

01 The transform families attackers use

The encodings fall into a few families, and each defeats a different naive filter:

- Encodings: Base64, Base32, Base58, hexadecimal, URL encoding. Turns readable words into strings a keyword filter does not recognise.
- Classical ciphers: ROT13, Caesar shifts, Morse code. Trivial for a model to reverse, invisible to a wordlist.
- Unicode look-alikes: homoglyphs, full-width characters, mathematical bold and italic alphabets, upside-down text. The characters render like normal letters to a human but carry different code points, so an exact-match filter misses them.
- Invisible text: zero-width characters and the Unicode Tag block let you hide an instruction inside text that a human reviewer sees as ordinary. The payload is literally not visible in a chat log or a document.
- Format tricks: character and word shuffling, spacing, and line reversal that a model can still parse.

A single request can stack several of these, which is why static rules struggle to keep up.

02 Why prompt obfuscation matters

Obfuscation is not a separate vulnerability. It is a delivery method that sits on top of prompt injection and jailbreaking, and it makes both harder to catch.

HOW TO DEFEND

- Canonicalise first. Apply Unicode NFKC normalisation, strip zero-width and Tag characters, and decode common encodings before any filter runs. Judge the normalised, decoded text, not the raw bytes.
- Classify intent, not keywords. A model-based check that reasons about what the input is asking for survives encodings that a wordlist cannot.
- Constrain the blast radius. Limit which tools an agent can call and what data it can reach, so a decoded instruction cannot do much even if it gets through.
- Red-team with the full transform set. Test your guardrail against encodings, ciphers, Unicode look-alikes, and invisible characters so you learn what it misses before an attacker does.

SOURCES

[1] OWASP LLM01:2025, Prompt Injection

[2] OWASP Top 10 for LLM Applications

[3] Unicode Normalization Forms (UAX #15)

[4] MITRE ATLAS, Adversarial ML tactics

Keyword blocklists and many classifier guardrails are trained on plaintext, so an encoded payload sails past them. Zero-width and Tag smuggling is worse: because the instruction is invisible, it rides inside indirect prompt injection through a support ticket, a web page, a PDF, or a code comment, and reaches a RAG pipeline or an agent without anyone seeing it in review. The same trick is used to bypass content moderation, where the model produces the blocked output because it decoded a request the filter never understood.

Test whether your AI guardrails survive obfuscated prompts.
securelayer7.net/learn/ai-security/prompt-obfuscation

Open online