SekaiCTF 2026: Filtered Reality — Solution

SekaiCTF 2026: Filtered Reality — Solution - dimasma0305, Project Sekai.

- Published: date not stated
- Original: <https://github.com/project-sekai-ctf/sekaictf-2026/blob/main/web/filtered-reality/solution/writeup.md>
- Preserved from: https://github.com/project-sekai-ctf/sekaictf-2026/blob/main/web/filtered-reality/solution/writeup.md (github-api) on 2026-09-13
- Licence: unknown

Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

writeup.md

`project-sekai-ctf/sekaictf-2026` at `main`, path `web/filtered-reality/solution/writeup.md`.

Filtered Reality — Author Writeup

A WordPress (6.6.2 / PHP 8.3) "records bureau" with a custom *Archive* plugin. The flag is held by a separate-UID **keeper** daemon and is never readable by the web user; an admin bot (the **Archivist**) reviews submitted records in a headless Chromium. The intended solve is a five-stage client server chain, each stage strictly necessary:

#	Stage	Primitive
0	Reviewer's Seal (screen-id spoof)	leak a screen-gated nonce to a subscriber and queue a record for the bot
1	SXG fallback-escape + DOM-clobber + trigram CSS nonce-leak	run JS in the Archivist (admin) under a per-reques
2	PHP object injection	a POP gadget through WordPress core to a file `include`
3	`php://filter` iconv chain	turn the include into RCE as `www-data`
4	keeper hash length-extension	forge a per-cycle token and read the flag

The public handout login is the subscriber `archive_clerk` / `ArchiveClerk!2026`.

Link 0 — Reviewer's Seal (the queue bootstrap)

The bot renders only records that have been *sealed* (queued). The queue's sole writer is the admin-ajax action `archive_seal_record`, which has **no capability check** — only a nonce. That nonce is printed by an `admin_enqueue_scripts` handler **only** when `get_current_screen()->id === 'toplevel_page_archive-desk'` (the privileged "Reviewer Desk"). A subscriber cannot load that page... but WordPress derives the current screen from `PHP_SELF`.

Requesting, as the logged-in clerk:

GET /wp-admin/index.php/x%0A/wp-admin/toplevel_page_archive-desk

the raw `%0A` opens a `PATH_INFO` break that WordPress folds back into `PHP_SELF`, so `$current_screen->id` resolves to `toplevel_page_archive-desk` on the dashboard a subscriber *can* load. The seal nonce is printed into the response; the clerk then POSTs `archive_seal_record` to queue any record. (Shared-container note: a `mu-plugin` makes WP sessions stateless so many teams can authenticate as the same clerk concurrently without the per-user `session_tokens` row racing.)

Link 1 — getting JS into the Archivist under a nonce CSP

`/?render=<ref>` echoes a record body verbatim, reflecting the request `Accept` into `Content-Type` (+`nosniff`) under `script-src 'nonce-X'`. A Chrome navigation `Accept` ends in `application/signed-exchange;v=b3`, so the body is parsed as a Signed HTTP Exchange and dies (`ERR_INVALID_SIGNED_EXCHANGE`) — a naïve `<script>` never runs.

Craft the body as an *invalid* SXG (`magic | fallbackUrlLen | https-fallback`). Chrome fails verification and performs the **SXG fallback navigation** to the embedded `https://` fallback — the internal `:1338` vhost serving `/?archive_moderation=1` — and that re-request's `Accept` is signed-exchange-free, so the page finally renders. The moderation page runs each record's `source` through a **clobberable** sanitizer. A DOM-clobbering `<form><input name=childNodes>...` smuggles a `<style>` whose `@import` pulls a same-origin hex-**trigram** CSS that leaks the per-request nonce `X` out of the (non-nonce-hidden) `<meta content>` to `/?archive_leak`. Reconstruct `X`, then clobber again to inject `<iframe srcdoc="<script nonce=X src=/?archive_raw=exp>">` — which runs in the admin context, scrapes the `archive_process_record` nonce, and POSTs the POP blob.

Link 2 — PHP object injection

`archive_process_record` (`manage_options` + nonce) does `@unserialize(base64_decode($blob))`. The intended gadget is the plugin's `SecureTableGenerator`: `__wakeup` sanitizes the `data` and `headers` cells (objects are neutralized there) — but **not** `allowedTags`. A non-empty `allowedTags` array survives `validateObjectIntegrity`, and `resetSecurityProperties()` then runs a **loose** `in_array($tag, $safeTags)` over it. Comparing a planted `WP_HTML_Tag_Processor` against a string invokes its `__toString()`, driving the WordPress-core POP chain (`WP_HTML_Tag_Processor` `WP_Block_List::offsetGet` `WP_Block::__construct` `WP_Block_Patterns_Registry::get_content`) into `include($filePath)`.

(The "obvious" `WP_HTML_Token::__destruct` gadget is a decoy: its `__wakeup` throws on 6.6.2, and on PHP 8.3 a throwing `__wakeup` means `__destruct` never runs.)

Link 3 — `php://filter` iconv RCE

`$filePath` is a `php://filter/convert.iconv.*|convert.base64-*/resource=php://temp` chain (synacktiv generator) that synthesises a small "keeper client" PHP payload from nothing and executes it on `include` — RCE as `www-data`. Because the include's output is captured by WP's output buffering, the payload writes its result to a `www-data`-writable

`wp-content/uploads/<t>.log` instead of relying on the response.

Link 4 — keeper hash length-extension

The flag is held by the `keeper` UID. Its daemon answers a local socket: `SIGN <msg>` returns `sha256(SECRET  msg)` but refuses any message containing `give_flag`; `FLAG <msg> <sig>` returns the flag iff the signature is valid and the message authorises `give_flag` **for** the **current cycle**. Sign a benign `cycle=<n>;user=guest`, then **length-extend** it to append `;cmd=give_flag`, producing a valid token **for** a message we were never allowed to sign. The keeper replies with the flag, which the RCE payload drops into `uploads/<t>.log`.

Running the solver

cd solution python3 solver.py --host <host> --port <port> # local: --host 127.0.0.1 --port 4000

`solver.py` (standard library only; helpers `filter_chain.py`, `build_blob.py`) drives all five links: clerk login `%0A` seal leak host the CSS/exploit/SXG records seal them reconstruct the nonce inject in-bot POP blob keeper forge read the flag **from** `uploads`.

Flag: `SEKAI{th3_d4y_n3v3r_3nds_1f_y0u_r34d_f4st}`
