April 21, 2026 · 18 min read · By Yiğit Konur

Warp cli-agent notification protokolünü reverse-engineer etmek

ai software-engineering

"Warp agent sidebar" dediğimiz şey sihir değil. Claude Code, Gemini CLI ya da OpenCode bir turu bitirdiğinde aslında olan şey şu: `/dev/tty`'ye tek bir escape sequence yazılıyor — JSON body'li tek bir OSC 777 çağrısı. Warp bunu pane'in output stream'inde yakalıyor, parse ediyor, sidebar'a yönlendiriyor.

bu işin public bir spec'i yok. ama Warp üç tane açık kaynak adapter ship'liyor — claude-code-warp (https://github.com/warpdotdev/claude-code-warp), gemini-cli-warp (https://github.com/warpdotdev/gemini-cli-warp), opencode-warp (https://github.com/warpdotdev/opencode-warp) — ve üçünü yan yana okuyunca tüm protokol gün yüzüne çıkıyor. altı envelope alanı, yedi event, tek bir transport primitive'i, tek bir feature flag.

bu post reverse-engineer edilmiş bir saha rehberi. buradaki her şey o üç repo'nun `b8ad3cc`, `953e05b` ve `e60e068` commit'lerinden gözlemlenmiş davranış. agent'ın tty üzerinden küçük bir parça JSON emit edebiliyorsa Warp onu da yakalıyor.

30 saniyelik özet

agent'lar Warp'a bildirim göndermek için `/dev/tty`'ye **tek bir OSC 777 escape sequence** yazıyor:

```
ESC ] 7 7 7 ; notify ; <TITLE> ; <BODY> BEL
```

ham string olarak (`\x1b` = ESC, `\x07` = BEL):

```
\x1b]777;notify;warp://cli-agent;<JSON-body>\x07
```

-

- **TITLE** literal olarak `warp://cli-agent` string'i. başka bir title verirsen plain-text notification çıkıyor (legacy yol).
- **BODY** compact bir JSON object.

Warp, pane'in tty'sinden gelen her OSC 777'yi parse ediyor. title `warp://cli-agent` ile eşleşirse body structured agent channel'a düşüyor. değilse notification center'da plain text olarak render ediliyor.

transport'un tamamı bu. gerisi sadece şema.

transport

`OSC 777`, desktop notification'lar için tarihten beri (gnome-terminal, urxvt) kullanılan eski bir xterm operating-system command'ı. Warp bunu kendine mal edip structured agent event'leri için private bir title namespace'i ekliyor.

herhangi bir dilden emit et:

bash

```bash
# POSIX shell
printf '\033]777;notify;%s;%s\007' "warp://cli-agent" "$JSON_BODY" > /dev/tty
```

typescript

```typescript
// Node / Bun
import { writeFileSync } from "fs";
const seq = `\x1b]777;notify;warp://cli-agent;${body}\x07`;
writeFileSync("/dev/tty", seq);
```

python

```python
# Python
with open("/dev/tty", "w") as tty:
    tty.write(f"\x1b]777;notify;warp://cli-agent;{body}\x07")
```

-

rust

```rust
// Rust
use std::fs::OpenOptions;
use std::io::Write;
let mut tty = OpenOptions::new().write(true).open("/dev/tty")?;
write!(tty, "\x1b]777;notify;warp://cli-agent;{}\x07", body)?;
```

neden /dev/tty

stdout da stderr de bu iş için yetmiyor.

agent host'larındaki hook script'lerinde **stdout genelde capture edilmiş** — structured return channel olarak kullanılıyor; host, event'i block'lamak, değiştirmek ya da log'lamak için stdout'tan JSON okuyor. stdout'a escape sequence yazarsan bu JSON bozuluyor.

stderr çoğu zaman log file'a pipe'lanıyor. kullanıcı görmüyor ve daha önemlisi escape terminale hiç ulaşmıyor.

`/dev/tty` process'in controlling terminal'i; pipe'ları bypass ediyor. write'lar pane'e tam olarak bir kez ulaşıyor. üç referans adapter de `/dev/tty`'ye yazıyor ve bir hata çıkarsa (örneğin controlling terminal yoksa) sessizce yutuyor. sen de aynısını yap.

SSH bedavaya geliyor

OSC 777 herhangi bir terminal byte'ı gibi pty üzerinden akıyor. Warp'tan bir makineye `ssh` at, orada agent çalıştır — notification'lar yine çalışıyor. Warp sequence'i yerel pty'ye ulaştığında görüyor. OpenCode adapter'ının `notify.ts` dosyasındaki "working over SSH" yorumu tam olarak bunu söylüyor: transport zaten terminal stream'i.

Warp receiving uçta değilse (örneğin remote server'a tmux'la girmişsin, dış terminal iTerm) bir şey bozulmuyor. escape'i anlamayan terminaller onu sessizce drop ediyor. ama kör körüne emit etme — böyle yaparsan plain-text pager'larda çöp görürsün. capability gate bunun için var.

-

capability gate

emit etmeden önce, karşı taraftaki terminal'in `warp://cli-agent`'i anlayan bir Warp build'i olduğundan emin ol. Warp desteği iki env var ile sinyalliyor:

`WARP_CLI_AGENT_PROTOCOL_VERSION`	`1`
`WARP_CLIENT_VERSION`	`v0.2026.04.15.08.24.stable_03`

üç adapter'ın da kullandığı tam bash gate:

-

```bash
# known-broken Warp release'leri (channel başına). bu build'ler
# WARP_CLI_AGENT_PROTOCOL_VERSION üzerinden protokol desteği
# advertise ediyor ama structured notification'ı aslında render
# etmiyor — feature, shipped binary'de enable olmayan bir flag
# arkasındaydı.
LAST_BROKEN_DEV=""
LAST_BROKEN_STABLE="v0.2026.03.25.08.24.stable_05"
LAST_BROKEN_PREVIEW="v0.2026.03.25.08.24.preview_05"

should_use_structured() {
    # protokol version advertise edilmemiş → Warp değil ya da çok eski.
    [ -z "${WARP_CLI_AGENT_PROTOCOL_VERSION:-}" ] && return 1

    # client version yok → broken build'leri eleyemiyoruz.
    [ -z "${WARP_CLIENT_VERSION:-}" ] && return 1

    # channel'a özel "broken floor" check.
    local threshold=""
    case "$WARP_CLIENT_VERSION" in
        *dev*)     threshold="$LAST_BROKEN_DEV" ;;
        *stable*)  threshold="$LAST_BROKEN_STABLE" ;;
        *preview*) threshold="$LAST_BROKEN_PREVIEW" ;;
    esac

    if [ -n "$threshold" ] && [[ ! "$WARP_CLIENT_VERSION" > "$threshold" ]]; then
        return 1
    fi

    return 0
}
```

`[[ ! X > Y ]]` bash'in lexicographic string karşılaştırması. Warp'un version string'leri lexicographic olarak sıralanabiliyor çünkü tarih component'leri dominant — o yüzden çalışıyor.

opencode-warp'taki sadeleştirilmiş TypeScript gate broken-build kontrolünü tamamen atlıyor:

-

typescript

```typescript
function warpNotify(title: string, body: string): void {
  if (!process.env.WARP_CLI_AGENT_PROTOCOL_VERSION) return;
  try {
    writeFileSync("/dev/tty", `\x1b]777;notify;${title};${body}\x07`);
  } catch {
    /* /dev/tty yok — yut */
  }
}
```

OpenCode adapter'ı broken stable release'ten sonra yazılmış, yani env var varlığı tek başına yeterli. yeni integration'larda sen de aynısını yapabilirsin **eğer** eski Warp build'indeki kullanıcıları umursamıyorsan — umursuyorsan yukarıdaki tam bash gate'i kopyala.

gate fail ettiğinde:

- **subprocess-hook modeli (Claude / Gemini):** `exit 0` ile sessizce çık ya da eski Warp versiyonları için plain-text OSC'ye fallback at.
- **in-process plugin'ler (OpenCode):** yazmadan dön. asla error verme — kullanıcı muhtemelen farklı bir terminaldedir.

payload envelope

her structured notification aynı altı alanlı envelope'u taşıyor. geri kalanı event'e özel.

json

```json
{
  "v": 1,
  "agent": "claude",
  "event": "prompt_submit",
  "session_id": "01J9K7P2E5S8V1Z3B2C4D6F8G0",
  "cwd": "/Users/alice/projects/my-app",
  "project": "my-app"
}
```

-

`v`			`1`
`agent`			`claude` `gemini` `opencode`
`event`			
`session_id`			
`cwd`			
`project`			`basename(cwd)`

version negotiation

```
negotiated_v = min(PLUGIN_MAX_PROTOCOL_VERSION, $WARP_CLI_AGENT_PROTOCOL_VERSION)
```

şu anda sadece `v=1` tanımlı. mekanizma, Warp v2 şemasına geçtiğinde eski adapter'lar hâlâ v1 konuşsun ve yeni adapter'lar eski Warp build'leriyle konuşurken aşağı negotiate etsin diye var.

>_ bash

```bash
PLUGIN_CURRENT_PROTOCOL_VERSION=1
negotiate_protocol_version() {
    local warp_version="${WARP_CLI_AGENT_PROTOCOL_VERSION:-1}"
    if [ "$warp_version" -lt "$PLUGIN_CURRENT_PROTOCOL_VERSION" ] 2>/dev/null; then
        echo "$warp_version"
    else
        echo "$PLUGIN_CURRENT_PROTOCOL_VERSION"
    fi
}
```

session binding implicit

pane ya da tab id'si gönderemezsin — Warp onu kendi yönetiyor:

1. OSC sequence `/dev/tty`'ye yazılıyor, o da *agent'ın çalıştığı pane'in* controlling terminal'i.
2. Warp o pane'in output stream'ini okuyor, yani sequence'i context içinde görüyor.
3. payload'daki `session_id`, pane'i temizlesen ya da birden fazla pencereye yayılsan bile aynı conversation'ın event'lerini gruplamasını sağlıyor.

yedi event

şu an canlı yedi event var. ilk altısını üç referans adapter de emit ediyor (host desteğine göre — aşağıdaki matrise bak). son ikisi (`question_asked`, `permission_replied`) OpenCode extension'ı.

session_start

agent yeni bir session başlattığında ya da eskisini resume ettiğinde bir kez tetikleniyor. Warp pane'i sidebar'a kaydediyor ve yüklü plugin'in güncel olup olmadığını kontrol ediyor.

```json
{
  "v": 1,
  "agent": "claude",
  "event": "session_start",
  "session_id": "01J9K7P2E5S8V1Z3B2C4D6F8G0",
  "cwd": "/Users/alice/projects/my-app",
  "project": "my-app",
  "plugin_version": "2.0.0"
}
```

ekstra alan: `plugin_version` (string) — adapter'ın kendi versiyonu. Warp bunu minimum required version'a karşı karşılaştırıyor ve altında kalırsa "outdated plugin" banner'ı gösteriyor.

-

prompt_submit

kullanıcı prompt submit ettiği an tetikleniyor. tab'ı **idle** / **done** → **running** durumuna geçiriyor.

```json
{
  "v": 1,
  "agent": "claude",
  "event": "prompt_submit",
  "session_id": "01J9K7P2E5S8V1Z3B2C4D6F8G0",
  "cwd": "/Users/alice/projects/my-app",
  "project": "my-app",
  "query": "refactor the auth middleware to use the new session store"
}
```

ekstra alan: `query` (string) — kullanıcının prompt'u, 200 karaktere truncate (daha uzunsa sonuna `...`). adapter'lar arası truncation kuralı tutarlı:

```bash
if [ -n "$QUERY" ] && [ ${#QUERY} -gt 200 ]; then
    QUERY="${QUERY:0:197}..."
fi
```

tool_complete

her tool call tamamlandığında tetikleniyor. tab'ı **blocked-on-tool** → **running** durumuna geçiriyor.

```json
{
  "v": 1,
  "agent": "claude",
  "event": "tool_complete",
  "session_id": "01J9K7P2E5S8V1Z3B2C4D6F8G0",
  "cwd": "/Users/alice/projects/my-app",
  "project": "my-app",
  "tool_name": "Bash"
}
```

-

ekstra alan: `tool_name` (string) — az önce çalışan tool'un tanımlayıcısı. serbest form; Warp sabit bir set belirlemiyor.

permission_request

protokolün en yüksek sesli event'i. agent, kullanıcı onayı gerektiren bir tool çalıştırmak istediğinde tetikleniyor. native OS notification tetikliyor ve tab'ı **blocked-awaiting-permission** olarak işaretliyor.

json

```
{
  "v": 1,
  "agent": "claude",
  "event": "permission_request",
  "session_id": "01J9K7P2E5S8V1Z3B2C4D6F8G0",
  "cwd": "/Users/alice/projects/my-app",
  "project": "my-app",
  "summary": "Wants to run Bash: rm -rf node_modules && npm install",
  "tool_name": "Bash",
  "tool_input": {
    "command": "rm -rf node_modules && npm install",
    "description": "Clean reinstall"
  }
}
```

ekstra alanlar:

`summary`		`Wants to run <tool>: <preview>`
`tool_name`		`tool_complete`
`tool_input`		`command` `file_path`

-

`on-permission-request.sh` 'ten summary-building heuristic'i:

```bash
TOOL_PREVIEW=$(echo "$INPUT" | jq -r '
    (.tool_input | if .command then .command
                   elif .file_path then .file_path
                   else (tostring | .[0:80]) end) // ""
')
SUMMARY="Wants to run $TOOL_NAME"
if [ -n "$TOOL_PREVIEW" ]; then
    if [ ${#TOOL_PREVIEW} -gt 120 ]; then
        TOOL_PREVIEW="${TOOL_PREVIEW:0:117}..."
    fi
    SUMMARY="$SUMMARY: $TOOL_PREVIEW"
fi
```

bu mantığı birebir kopyala — yoksa notification'ların "Wants to run Bash" diyor, komutun ne olduğuna dair en ufak ipucu olmadan.

idle_prompt

agent yeterince uzun süre idle kaldığında — muhtemelen input bekliyordur diye — tetikleniyor. `event` değeri host'un notification system'i ne diyorsa o oluyor, en sık `idle_prompt`, o yüzden hardcode etme, aynen geçir.

```json
{
  "v": 1,
  "agent": "claude",
  "event": "idle_prompt",
  "session_id": "01J9K7P2E5S8V1Z3B2C4D6F8G0",
  "cwd": "/Users/alice/projects/my-app",
  "project": "my-app",
  "summary": "Claude is waiting for your input"
}
```

ekstra alan: `summary` (string) — native notification'da gösterilen serbest-form mesaj. host vermemişse default olarak `Input needed`.

stop

agent bir turu bitirdiğinde tetikleniyor — session sonu değil, sadece "şu an için konuşma bitti". tab **done** durumuna geçiyor ve son prompt/response çifti ile native notification raise ediyor.

json

```
{
  "v": 1,
  "agent": "claude",
  "event": "stop",
  "session_id": "01J9K7P2E5S8V1Z3B2C4D6F8G0",
  "cwd": "/Users/alice/projects/my-app",
  "project": "my-app",
  "query": "refactor the auth middleware to use the new session store",
  "response": "I refactored `middleware/auth.ts` to use `SessionStore.get()` and updat
  "transcript_path": "/Users/alice/.claude/projects/my-app/conversation-01J9K7P2.jsonl
}
```

ekstra alanlar:

`query`	
`response`	
`transcrip` `t_path`	

stop-hook race. Claude Code'a özel detay ama transcript'i async yazan tüm host'lar için geçerli:

- Claude Code `Stop`'u transcript file flush olmadan *önce* tetikliyor. adapter 0.3s uyuyor, sonra `jq` ile son user + assistant mesajlarını okuyor.
- host `stop_hook_active` flag'i veriyorsa mutlaka kontrol et — stop event'i replay edildiğinde (örneğin recovery sonrası) `true` oluyor, double notification'ı önlüyor.

```bash
STOP_HOOK_ACTIVE=$(echo "$INPUT" | jq -r '.stop_hook_active // false')
[ "$STOP_HOOK_ACTIVE" = "true" ] && exit 0

sleep 0.3  # transcript flush olsun

TRANSCRIPT_PATH=$(echo "$INPUT" | jq -r '.transcript_path // empty')
# ... JSONL'den son user + assistant mesajlarını oku
```

question_asked (OpenCode extension)

OpenCode'un yerleşik bir `question` tool'u var, agent clarifying soru sormak için kullanıyor. çağrıldığında adapter bu event'i gönderiyor, Warp "input lazım" durumunu generic tool call'dan ayırabiliyor.

```json
{
  "v": 1,
  "agent": "opencode",
  "event": "question_asked",
  "session_id": "sess_01J9K7P2E5S8V1Z3B2C4D6F8G0",
  "cwd": "/Users/alice/projects/my-app",
  "project": "my-app",
  "tool_name": "question"
}
```

benzer bir meta-tool pattern'i olan agent yazıyorsan bu event adını tekrar kullan — Warp'ta UI zaten bağlı.

permission_replied (OpenCode extension)

kullanıcı permission request'e cevap verdiğinde ve reddetmediğinde tetikleniyor. Warp "awaiting permission" state'ini preemptively temizleyebiliyor, takip eden `tool_complete`'i beklemeden.

-

```json
{
  "v": 1,
  "agent": "opencode",
  "event": "permission_replied",
  "session_id": "sess_01J9K7P2E5S8V1Z3B2C4D6F8G0",
  "cwd": "/Users/alice/projects/my-app",
  "project": "my-app"
}
```

sadece kullanıcı **izin verdiğinde** emit et. reject'ler zaten ya `stop` ya da başka bir `permission_request` ile devam ediyor.

iki integration shape'i

üç resmi adapter iki integration shape'i gösteriyor: subprocess-hook (bash) ve in-process plugin (TypeScript). host'unun extension modeline uyanı seç.

subprocess hook'lar (claude-code-warp, gemini-cli-warp)

host CLI'da her lifecycle event'inin bir external command'ı tetiklediği bir hook sistemi var, event verisini JSON olarak stdin'den geçiriyor. command:

1. stdin'i okuyor.
2. host'un davranışını etkilemek için (örneğin bir tool call'u block'lamak için) opsiyonel olarak stdout'a structured JSON emit ediyor.
3. side effect emit ediyor — bizim durumumuzda `/dev/tty`'ye bir OSC 777.

registration adapter'a check'lenmiş bir JSON file. Claude Code formatı:

-

json

```
{
  "description": "Warp terminal notifications",
  "hooks": {
    "SessionStart": [
      {
        "matcher": "startup|resume",
        "hooks": [
          { "type": "command",
            "command": "${CLAUDE_PLUGIN_ROOT}/scripts/on-session-start.sh" }
        ]
      }
    ],
    "UserPromptSubmit": [
      { "hooks": [
        { "type": "command",
          "command": "${CLAUDE_PLUGIN_ROOT}/scripts/on-prompt-submit.sh" }
      ]}
    ],
    "PostToolUse": [
      { "hooks": [
        { "type": "command",
          "command": "${CLAUDE_PLUGIN_ROOT}/scripts/on-post-tool-use.sh" }
      ]}
    ],
    "PermissionRequest": [
      { "hooks": [
        { "type": "command",
          "command": "${CLAUDE_PLUGIN_ROOT}/scripts/on-permission-request.sh" }
      ]}
    ],
    "Notification": [
```

Gemini'nin formatı neredeyse aynı, sadece event adları farklı: `SessionStart`, `BeforeAgent`, `AfterTool`, `Notification`, `AfterAgent`.

event başına script iskeleti:

-

```bash
#!/bin/bash
# on-prompt-submit.sh
SCRIPT_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
source "$SCRIPT_DIR/should-use-structured.sh"

if ! should_use_structured; then
    exit 0
fi

source "$SCRIPT_DIR/build-payload.sh"

INPUT=$(cat)                                          # hook input stdin'den
QUERY=$(echo "$INPUT" | jq -r '.prompt // empty')
[ ${#QUERY} -gt 200 ] && QUERY="${QUERY:0:197}..."

BODY=$(build_payload "$INPUT" "prompt_submit" \
    --arg query "$QUERY")

"$SCRIPT_DIR/warp-notify.sh" "warp://cli-agent" "$BODY"
```

envelope factory (`build-payload.sh`):

-

```bash
PLUGIN_CURRENT_PROTOCOL_VERSION=1

negotiate_protocol_version() {
    local warp_version="${WARP_CLI_AGENT_PROTOCOL_VERSION:-1}"
    if [ "$warp_version" -lt "$PLUGIN_CURRENT_PROTOCOL_VERSION" ] 2>/dev/null; then
        echo "$warp_version"
    else
        echo "$PLUGIN_CURRENT_PROTOCOL_VERSION"
    fi
}

build_payload() {
    local input="$1"
    local event="$2"
    shift 2

    local protocol_version session_id cwd project
    protocol_version=$(negotiate_protocol_version)
    session_id=$(echo "$input" | jq -r '.session_id // empty')
    cwd=$(echo "$input" | jq -r '.cwd // empty')
    project=""
    [ -n "$cwd" ] && project=$(basename "$cwd")

    jq -nc \
        --argjson v "$protocol_version" \
        --arg agent "myagent" \
        --arg event "$event" \
        --arg session_id "$session_id" \
        --arg cwd "$cwd" \
        --arg project "$project" \
        "$@" \
```

transport (`warp-notify.sh`):

```bash
#!/bin/bash
SCRIPT_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
source "$SCRIPT_DIR/should-use-structured.sh"
should_use_structured || exit 0

TITLE="${1:-Notification}"
BODY="${2:-}"
printf '\033]777;notify;%s;%s\007' "$TITLE" "$BODY" > /dev/tty 2>/dev/null || true
```

in-process plugin (opencode-warp)

host CLI bir plugin API'si açıyor (TypeScript callback'leri, Go plugin interface'leri, Python entry point'ler...). event handler'ları kaydediyorsun, host'un kendi process'inde çalışıyor ve OSC sequence oradan yazılıyor.

TypeScript taslağı:

```typescript
import { writeFileSync } from "fs";
import path from "path";

const PLUGIN_VERSION = "0.1.0";
const PLUGIN_MAX_PROTOCOL_VERSION = 1;
const NOTIFICATION_TITLE = "warp://cli-agent";

function negotiateV(): number {
  const w = parseInt(process.env.WARP_CLI_AGENT_PROTOCOL_VERSION ?? "1", 10);
  return isNaN(w) ? PLUGIN_MAX_PROTOCOL_VERSION : Math.min(w, PLUGIN_MAX_PROTOCOL_VERS
}

function buildPayload(
  event: string,
  sessionId: string,
  cwd: string,
  extra: Record<string, unknown> = {},
): string {
  return JSON.stringify({
    v: negotiateV(),
    agent: "myagent",
    event,
    session_id: sessionId,
    cwd,
    project: cwd ? path.basename(cwd) : "",
    ...extra,
  });
}

function warpNotify(body: string): void {
  if (!process.env.WARP_CLI_AGENT_PROTOCOL_VERSION) return;
```

uyumluluk matrisi

hangi host event'i hangi structured event'e map oluyor — ve kim neyi destekliyor.

-

`session_start`	`SessionStart` `startup|resume`	`SessionStart` `startup`	`session.created`
`prompt_submit`	`UserPromptSubmit`	`BeforeAgent`	`chat.message`
`tool_complete`	`PostToolUse`	`AfterTool`	`tool.execute.after`
`permission_request`	`PermissionRequest`	`Notification` `notification_type=ToolPermission`	`permission.updated` `permission.asked`
`idle_prompt`	`Notification` `idle_prompt`	`Notification`	
`stop`	`Stop`	`AfterAgent`	`session.idle`
`question_asked`			`tool.execute.before`
`permission_replied`			`permission.replied`

legacy fallback

structured notification'lardan önceki Warp build'leri için Claude Code adapter'ı paralel bir `legacy/*.sh` tree'si bulunduruyor, human-readable title ve body ile plain-text OSC 777 notification emit ediyor.

```bash
printf '\033]777;notify;%s;%s\007' "Claude Code" "Task complete: $RESPONSE" > /dev/tty
```

bunlar Warp'ın notification center'ında sidebar entegrasyonu olmayan generic text olarak görünüyor. dispatch pattern:

```bash
if ! should_use_structured; then
    [ "$TERM_PROGRAM" = "WarpTerminal" ] && exec "$SCRIPT_DIR/legacy/on-stop.sh"
    exit 0
fi
```

yeni integration'lar legacy tree'yi tamamen atlayabilir. stable-channel broken build bu yazı yazıldığında bir yaşında ve çoğu kullanıcı güncel.

seni ısıran edge case'ler

jq hard requirement (bash adapter'lar için)

tüm bash adapter'lar payload'ı `jq -nc` ile kuruyor — doğru JSON escaping için. `jq` yoksa Claude Code `SessionStart` hook'u görünür bir `systemMessage` ile kullanıcıya kurmasını söylüyor:

```bash
if ! command -v jq &>/dev/null; then
    cat << 'EOF'
{"systemMessage": "Warp notifications require jq! Install it with brew install jq"}
EOF
    exit 0
fi
```

`printf` ile JSON'u elle kurmaya kalkışma — payload'lar naif escaping'i anında kıran kullanıcı verisi (komut, file path, prompt) taşıyor.

stdout'un sessiz kaldığına güvenme

subprocess-hook modelinde script'inin stdout'a yazdığı her şey host CLI tarafından interpret ediliyor. stdout'a debug log basarsan control JSON olarak parse edilip host'u

bozabilir ya da kafa karıştırıcı davranışa yol açabilir. stderr'e ya da dosyaya log bas.

Gemini CLI adapter'ı özellikle bu konuda paranoyak — her script `echo '{}'` ile bitiyor, host'a well-formed boş JSON objesi verip sessizliği yanlış yorumlamasını engelliyor:

```bash
# ... notification gönder ...
echo '{}'  # "müdahale yok" sinyali
```

message.updated defalarca tetikleniyor — filtrele

OpenCode'un `message.updated` event'i her partial token stream update'inde tetikleniyor, mesaj başına bir kez değil. bunu `prompt_submit` tetikleyicisi olarak kullanırsan onlarca duplicate üretiyor ve geç gelen biri `stop` notification'ını eziyor. onun yerine host'un sunduğu "message complete" / "user message finalized" sinyali ne ise onu kullan — OpenCode'da `chat.message`.

host'unda öyle bir sinyal yoksa `session_id` + monoton bir message counter ile debounce et.

tty olmayan context'lerden emit etme

agent CI içinde, pty'siz subprocess'te ya da Warp olmayan bir terminalde çalışıyorsa `/dev/tty`'ye yazmak ya fail eder ya da stream'i okuyan parent'ın çıktısını bozar. env-var gate'i (`WARP_CLI_AGENT_PROTOCOL_VERSION`) kritik kontrol. write'ın etrafındaki `try { ... } catch {}` sadece safety net, filter değil.

session-id stability

Warp event'leri `session_id` ile correlate ediyor. host bunu conversation ortasında yeniden üretirse (örneğin resume'da) Warp bunu yeni bir session olarak algılıyor ve yeni bir sidebar entry açıyor. host'un canonical session id'sini (ULID, UUID, vs.) kullan, kendinkini icat etme.

-

protocol version: aşağı yuvarla, yukarı değil

Warp `v=2` advertise ediyor ve adapter'ın sadece `v=1` biliyorsa `v=1` emit et. Warp `v=1`'i sonsuza kadar (en azından deprecation window boyunca) parse etmek zorunda. kendi üretmediğin bir version'ı asla emit etme, Warp yüksek destek verdiğini söylese bile.

debugging

ne emit ettiğini gör

script'inin tty write'larını geçici olarak bir dosyaya pipe'la:

```bash
# önce:
printf '\033]777;notify;%s;%s\007' "$TITLE" "$BODY" > /dev/tty

# debug sırasında:
printf '\033]777;notify;%s;%s\007' "$TITLE" "$BODY" | tee -a /tmp/warp-osc.log > /dev/
```

sonra `cat -v /tmp/warp-osc.log` ile escape sequence'leri insan-okunur formda görebilirsin.

emit etmeden önce JSON'u validate et

```bash
BODY=$(build_payload "$INPUT" "stop" --arg query "$QUERY")
echo "$BODY" | jq . >/dev/null 2>&1 || {
    echo "[warp-adapter] malformed body: $BODY" >&2
    exit 0
}
```

body'deki malformed JSON Warp'ın notification'ı sessizce drop etmesine yol açıyor. hiçbir yerde error görmüyorsun.

Warp olmadan test et

env var'ları normal bir terminalde elle set et:

```bash
export WARP_CLI_AGENT_PROTOCOL_VERSION=1
export WARP_CLIENT_VERSION="v0.2026.04.21.08.24.stable_01"
```

şimdi `should_use_structured` true dönüyor ve script'lerin full code path'ini çalıştırıyor. OSC sequence terminalde çöp olarak yazılıyor — amaç bu, `cat -v` ile bakabilirsin. unit test için iyi.

end-to-end test harness

her iki bash adapter'ı da bir `tests/test-hooks.sh` ship'liyor, sample Claude / Gemini hook input'larıyla stdin'i stub'lıyor ve emit edilen byte'ları assert ediyor. okumaya değer:

- `warpdotdev/claude-code-warp/tests/test-hooks.sh`
- `warpdotdev/gemini-cli-warp/tests/test-hooks.sh`

OpenCode adapter'ının `tests/*.test.ts` altında düzgün bir vitest suite'i var.

tam bir reference implementation

her lifecycle event'inde shell command çalıştırmana izin veren bir host'a bunu at. `AGENT_SLUG`'ı değiştir, event handler'ları bağla, bitti.

-

bash

```bash
#!/bin/bash
# warp-adapter/warp-notify.sh
set -euo pipefail

# ----- config -----
AGENT_SLUG="myagent"
PLUGIN_VERSION="1.0.0"
PLUGIN_MAX_PROTOCOL_VERSION=1

LAST_BROKEN_STABLE="v0.2026.03.25.08.24.stable_05"
LAST_BROKEN_PREVIEW="v0.2026.03.25.08.24.preview_05"

# ----- gate -----
should_use_structured() {
    [ -z "${WARP_CLI_AGENT_PROTOCOL_VERSION:-}" ] && return 1
    [ -z "${WARP_CLIENT_VERSION:-}" ] && return 1
    local threshold=""
    case "$WARP_CLIENT_VERSION" in
        *stable*)  threshold="$LAST_BROKEN_STABLE" ;;
        *preview*) threshold="$LAST_BROKEN_PREVIEW" ;;
    esac
    if [ -n "$threshold" ] && [[ ! "$WARP_CLIENT_VERSION" > "$threshold" ]]; then
        return 1
    fi
    return 0
}

# ----- payload -----
negotiate_v() {
    local w="${WARP_CLI_AGENT_PROTOCOL_VERSION:-1}"
    if [ "$w" -lt "$PLUGIN_MAX_PROTOCOL_VERSION" ] 2>/dev/null; then
```

kullan:

bash

```bash
source ./warp-adapter/warp-notify.sh

SESSION="sess_$(uuidgen)"
warp_session_start "$SESSION" "$PWD"
warp_prompt_submit "$SESSION" "$PWD" "refactor the retry loop"
warp_tool_complete "$SESSION" "$PWD" "Edit"
warp_permission_request "$SESSION" "$PWD" "Bash" "rm -rf node_modules" '{"command":"rm
warp_stop "$SESSION" "$PWD" "refactor the retry loop" "Done, tests pass." "/tmp/transc
```

-

tldr

`warp://cli-agent` protokolü bilinçli olarak küçük. altı envelope alanı, yedi event, tek bir transport primitive'i, tek bir feature flag. portable olmasını sağlayan bu minimalizm — üç referans implementation'ı, herhangi bir dilde 25 satırın altında bir transport helper paylaşıyor.

yeni bir agent'a Warp desteği eklemek neredeyse tamamen **host adapter layer'ında** iş — CLI'nın native lifecycle event'lerini yukarıdaki yedi structured event'e map etmek. protokolün kendisi bir öğleden sonrada bitiyor.

protokol evrilirken bakılacak üç şey:

- **protokol v2.** negotiation mekanizması hazır ama şu an sadece v1 canlı. Warp bir gün bump ederse rework yerine envelope'a yeni opsiyonel alanlar (cost, token count, structured error code) beklemek daha doğru.
- **yeni event'ler.** `question_asked` ve `permission_replied` OpenCode extension'ı olarak başladı, core set'e terfi edebilir. host'unda "clarifying question" konsepti varsa `question_asked`'i şimdiden emit et.
- **outdated-plugin banner'ı.** Warp, `session_start`'taki `plugin_version`'u hardcoded bir floor'a karşı karşılaştırıyor. adapter'ın breaking change ship'lediğinde version'u bump et *ve* Warp ile koordine et — `MINIMUM_PLUGIN_VERSION` constant'ını güncellesinler.

üç adapter repo'su da MIT lisanslı. ihtiyacın olan parçaları kopyala.

SOURCES

warpdotdev/claude-code-warp (https://github.com/warpdotdev/claude-code-warp)

warpdotdev/gemini-cli-warp (https://github.com/warpdotdev/gemini-cli-warp)

warpdotdev/opencode-warp (https://github.com/warpdotdev/opencode-warp)

Warp notifications docs (https://docs.warp.dev/features/notifications)

xterm control sequences (OSC reference) (https://invisible-island.net/xterm/ctlseqs/ctlseqs.html)