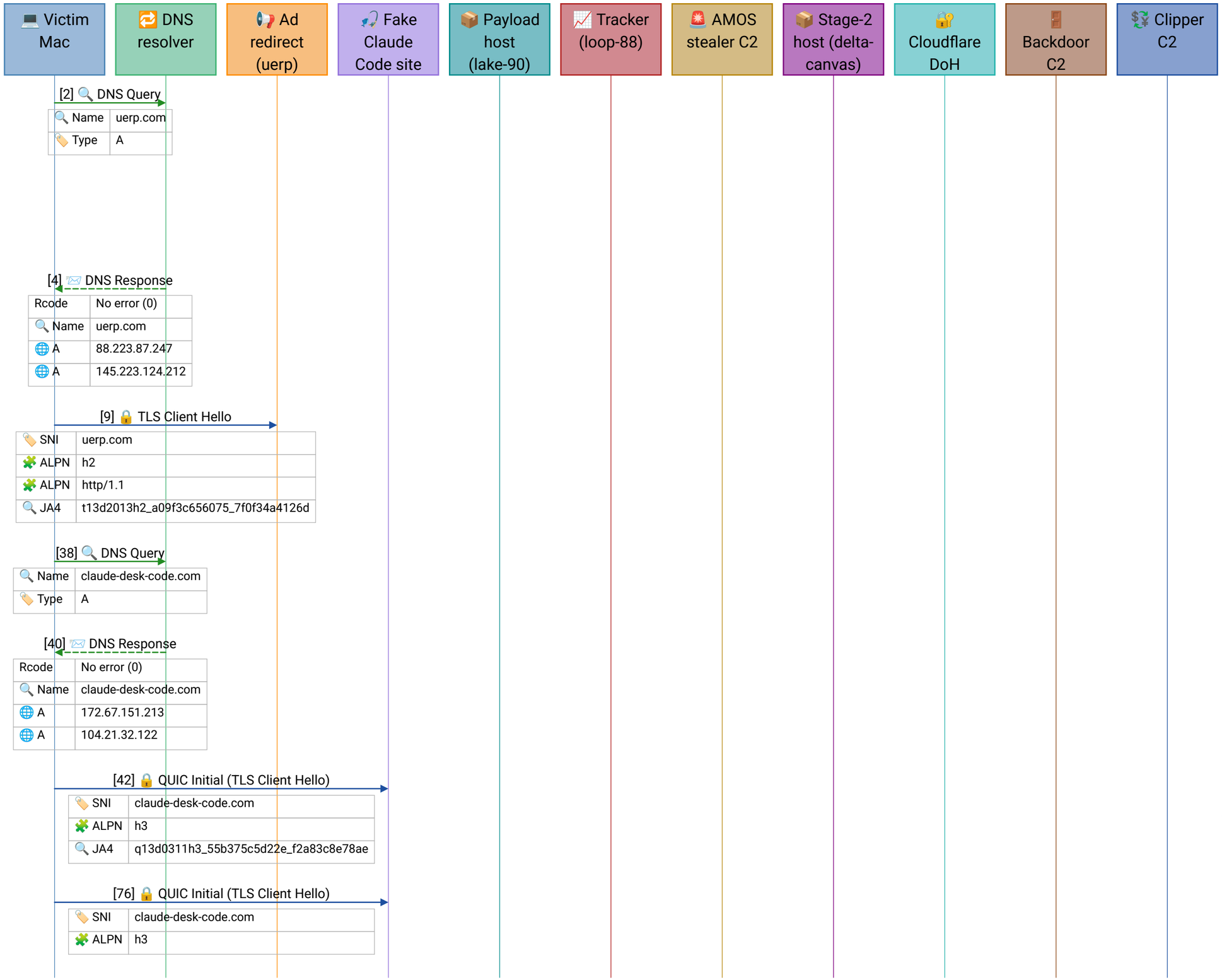


[Frame 2]

Phase 1: A search ad leads to a fake Claude Code site

The Mac's first lookup in the capture is for uerp [.]com, the domain behind the Google ad Brad clicked at 22:25 UTC. The ad shows a trusted-looking listing, but its click-through URL passes through this redirect first.

Safari also asks for an HTTPS -type record alongside each name; the diagram keeps only the A (IPv4 address) lookups.

[Frame 4] The local resolver answers with two IPv4 addresses for uerp [.]com. Nothing here is unusual yet: a single DNS answer cannot tell a redirector from an ordinary site.

[Frame 9] Safari opens HTTPS to the redirector. The contents are encrypted, but the TLS **Client Hello** is not: it names the site in the `SNI` field and carries the browser's fingerprint.

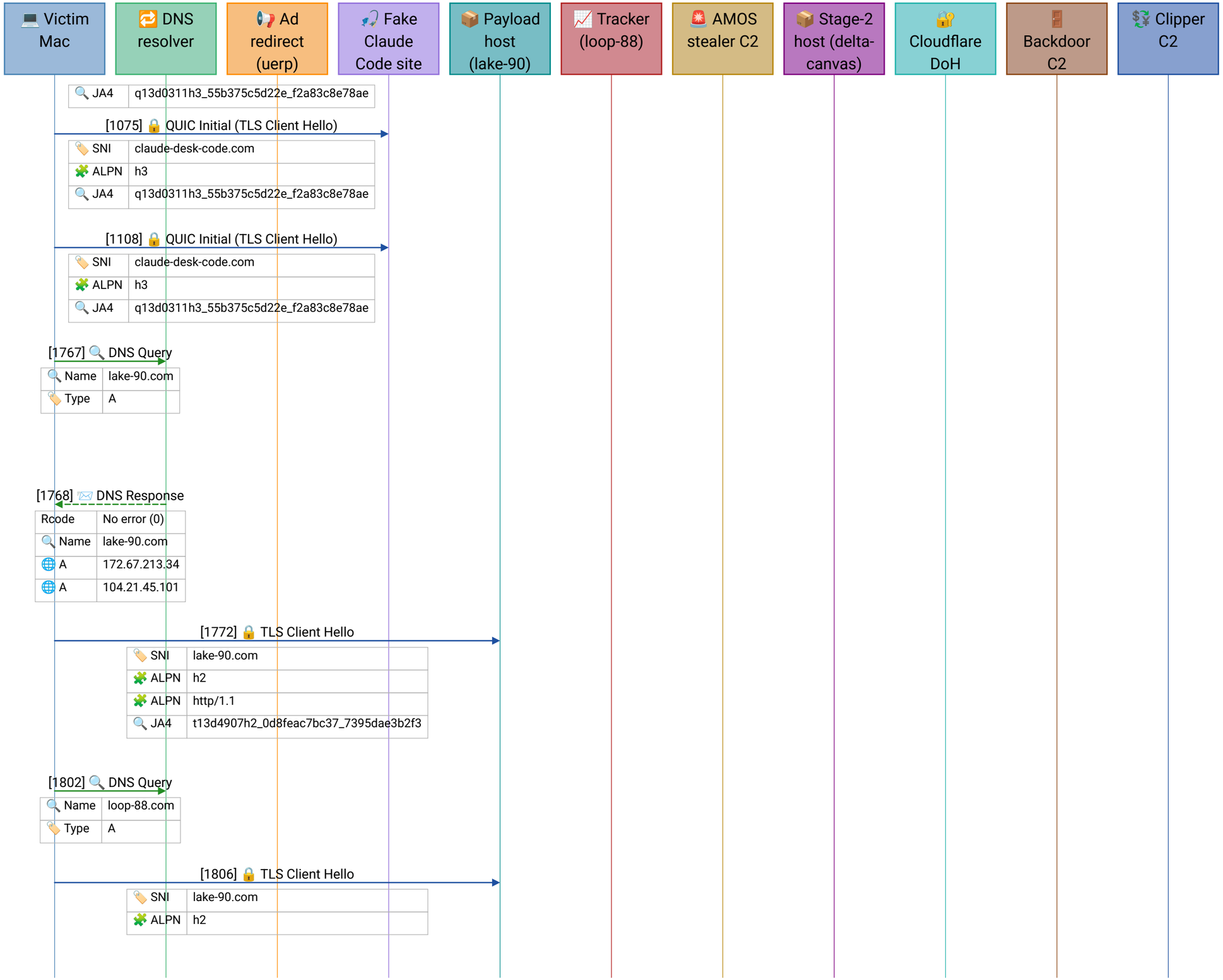
Remember this JA4 value, `t13d2013h2_....` It is Safari's. Later, a different fingerprint will show that something other than the browser is talking.

[Frame 38] The redirect sends Safari to claude-desk-code [.]com, a look-alike domain hosting a page that impersonates the Claude Code install page. This is the lookup that a DNS-based blocklist would have to catch.

[Frame 40] Two addresses come back, 172.67.151 [.]213 and 104.21.32 [.]122. Both belong to Cloudflare, which fronts the fake site, so blocking by IP address would also block many legitimate sites behind the same provider. The domain name is the useful indicator.

[Frame 42] Safari reaches the fake site over QUIC (HTTP/3, UDP port 443). The first QUIC packet, the Initial, is encrypted only with keys anyone can derive, so Wireshark can read the Client Hello inside it: `SNI` claude-desk-code [.]com, `ALPN` `h3`, and Safari's QUIC fingerprint `q13d0311h3_....`

[Frame 76] A second QUIC connection to the same site, a third of a second later: Safari loading the rest of the page.



[Frame 1075] Fifteen seconds later Safari connects to the fake site again. This fits the visitor moving through the page. Brad's screenshots show it presenting "install" instructions: copy a one-line command and paste it into Terminal. This is the **ClickFix** trick: the victim runs the malware themselves, so no download warning or Gatekeeper check ever fires.

[Frame 1108] The same page, now reached through the site's other Cloudflare address. Safari's part in the infection ends here. Everything from frame 1767 on is made by the pasted command and the programs it starts.

[Frame 1767]

Phase 2: The pasted command runs

82 seconds after Safari's last connection to the fake site, the Mac looks up lake-90 [.]com. That name appears nowhere on the page: it was hidden inside the pasted command as a base64 string, decoded by `openssl` and passed to `curl`, whose output went straight into `zsh`.

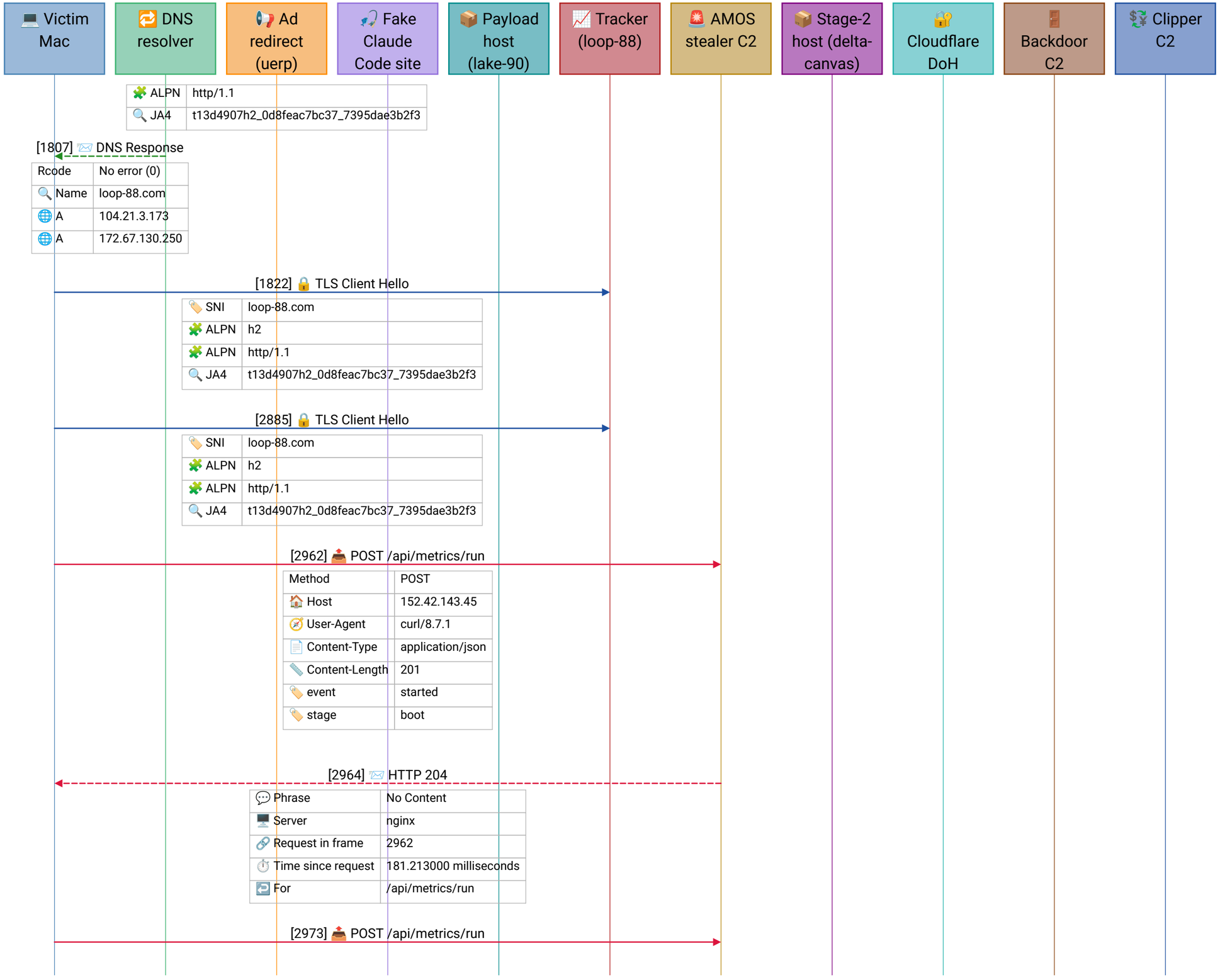
[Frame 1768] lake-90 [.]com resolves to two more Cloudflare addresses. The attackers spread their infrastructure across several throwaway domains behind the same provider.

[Frame 1772] Here is the tell. This Client Hello carries a different fingerprint, `t13d4907h2_...`, not Safari's `t13d2013h2_...` from frame 9. It is macOS `curl`, the same program the plain-HTTP requests below identify in their `User-Agent`.

This connection fetches the first-stage script, a 3,106-byte `zsh` file (per Brad's notes); about 8.5 kB of TLS records come back, consistent with that.

[Frame 1802] The script looks up a second domain, loop-88 [.]com. Its job, from Brad's notes, is telemetry: `curl` sends `POST` requests to `/api/metrics/run?event=pasted` and later `?event=stager`, telling the operators that a victim pasted the command and that the stager ran.

[Frame 1806] A second `curl` connection to lake-90 [.]com. This one downloads about 1.2 MB, which matches the 1,167,232-byte stealer binary Brad recovered at `/tmp/.zxqa03cu`: a universal Mach-O file for both Intel and Apple-silicon Macs.



[Frame 1807] loop-88 [.]com resolves to Cloudflare addresses as well.

[Frame 1822] `curl` reports to the tracker. Same `curl` fingerprint; the request itself (`event=pasted`) is inside TLS, so only Brad's notes show its path.

[Frame 2885] The second tracker report, `event=stager`, a second after the first. From here on the stealer talks to its own server over plain HTTP, and every byte is readable.

[Frame 2962]

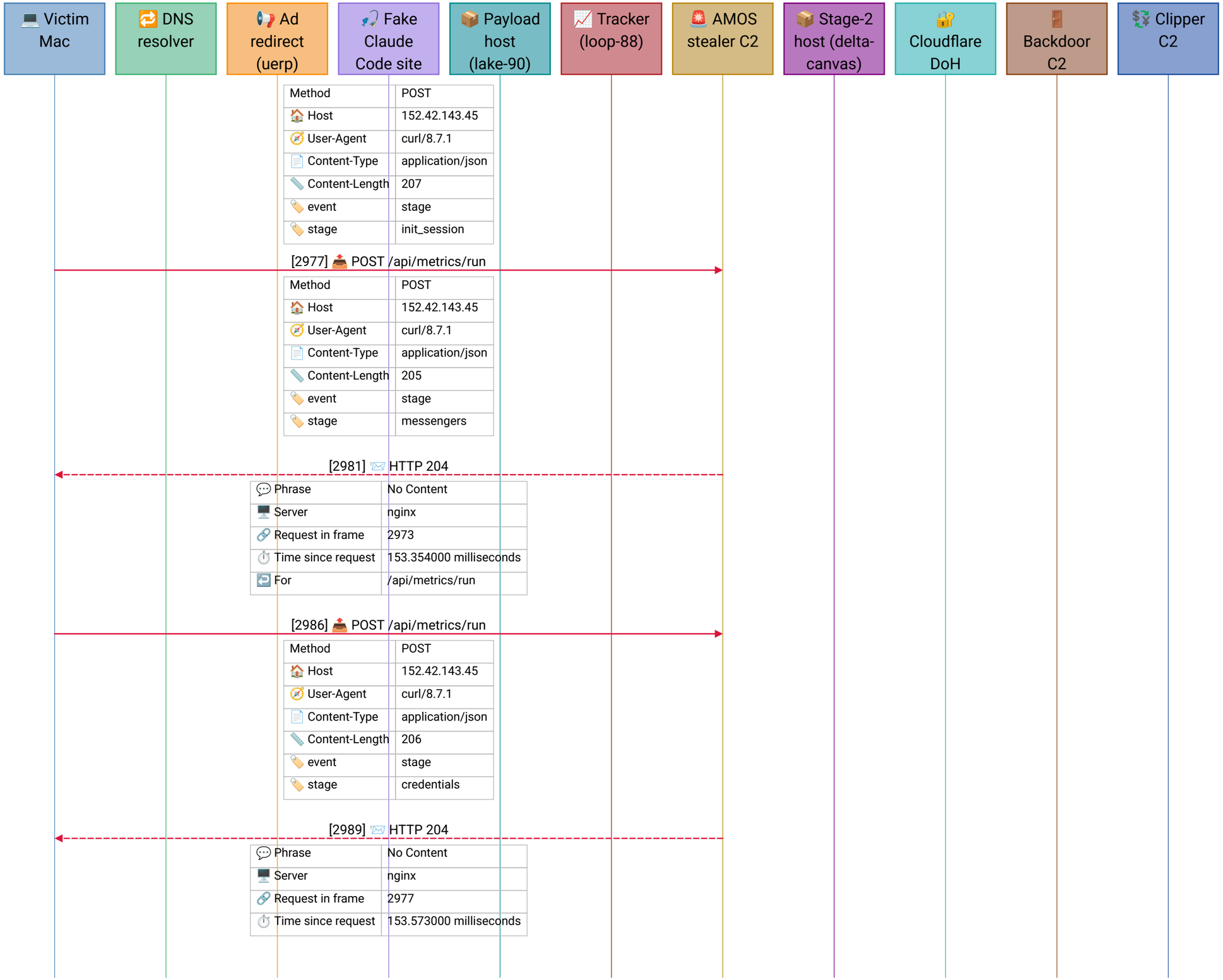
Phase 3: The stealer checks in and starts collecting

The stealer's first message to its command-and-control (C2) server at 152.42.143 [.]45: a JSON POST over plain HTTP on port 80, by bare IP address with no DNS lookup.

The body describes the victim (`macOS`, `27.0.1`, `arm64`, `en_US`), a victim ID `u`, a build ID `b`, and `event: started`, `stage: boot`. Three red flags in one packet: `curl` as the `User-Agent`, a raw IP as the `Host`, and data posted to a server nobody looked up.

[Frame 2964] The server replies `204 No Content`: received, nothing to say. It answers every progress report this way.

[Frame 2973] `stage: init_session`. The stealer reports each step as it starts, so the operators can see how far each



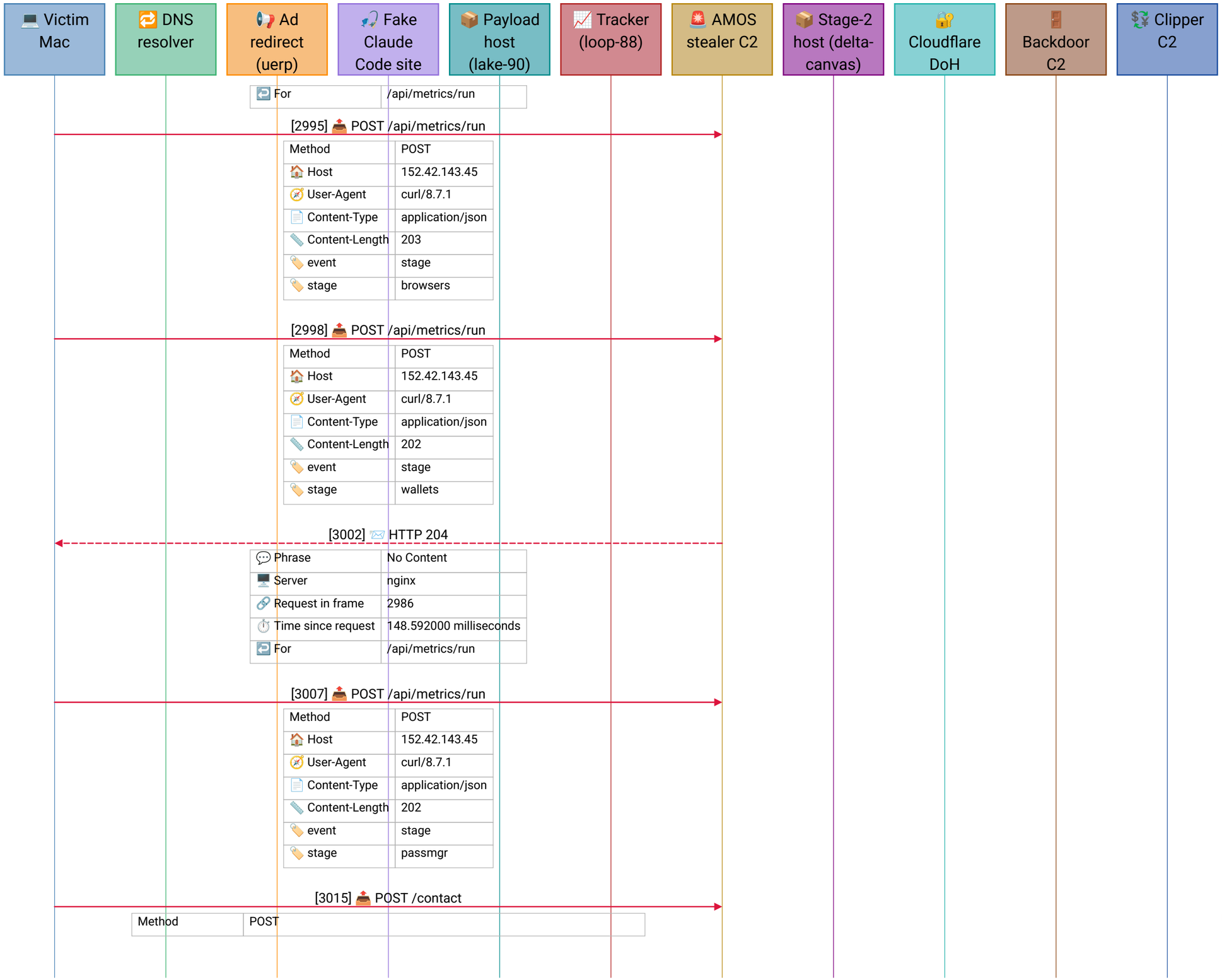
infection gets. Each report opens a new TCP connection.

[Frame 2977] `stage: messengers:` it searches for chat apps such as Telegram, whose local data can be used to take over the account.

[Frame 2981] `204` for the `init_session` report (frame 2973). Replies arrive out of order because each report travels on its own connection.

[Frame 2986] `stage: credentials:` cloud and developer credentials. Brad's file list shows it searched for AWS, Google Cloud, Docker, and FileZilla configuration, plus the shell history.

[Frame 2989] `204` for the `messengers` report (frame 2977).



[Frame 2995] `stage:browsers:` saved passwords, cookies, and autofill data from installed browsers.

[Frame 2998] `stage:wallets:` desktop cryptocurrency wallets. The file list shows folders for Binance and TonKeeper.

[Frame 3002] `204` for the `credentials` report (frame 2986).

[Frame 3007] `stage:passmgr:` password managers. The first upload, next, includes a `passmgr/Bitwarden` folder.

[Frame 3015] The first theft, 0.8 seconds after boot. A `multipart/form-data` upload to `/contact` carries the victim ID `u`, build ID `b`, three one-digit flags `(l, n, p)`, and a 3,249-byte zip named like a hidden macOS temp file.

💻 Victim Mac | 🔁 DNS resolver | 📢 Ad redirect (uerp) | 🎣 Fake Claude Code site | 📦 Payload host (lake-90) | 📈 Tracker (loop-88) | 🚨 AMOS stealer C2 | 📦 Stage-2 host (delta-canvas) | 🔐 Cloudflare DoH | 🚪 Backdoor C2 | 💱 Clipper C2

🏠 Host	152.42.143.45
🧭 User-Agent	curl/8.7.1
📏 Content-Length	4069
📄 Content-Type	multipart/form-data; boundary=----------7JvFidNkO9sR2yj2cmeXgy
📎 Form field	u
📎 Form field	b
📎 Form field	l
📎 Form field	n
📎 Form field	p
🗜️ File part	.A86Eo_A86ED3CE-61F0-40E0-A4F3-6F85D42B88DA.zip

Exporting the zip from the capture shows what it holds so far: an `info` file (1,338 bytes), the `username`, the shell history, and empty folders for the apps it searched. The most valuable item, the user's password, is still missing; see frame 3157.

[3017] 📨 HTTP 204

💬 Phrase	No Content
🖥️ Server	nginx
🔗 Request in frame	2995
⏱️ Time since request	147.274000 milliseconds
↩️ For	/api/metrics/run

[Frame 3017] `204` for the `browsers` report (frame 2995).

[3021] 📨 HTTP 204

💬 Phrase	No Content
🖥️ Server	nginx
🔗 Request in frame	2998
⏱️ Time since request	149.693000 milliseconds
↩️ For	/api/metrics/run

[Frame 3021] `204` for the `wallets` report (frame 2998).

[3027] 📨 HTTP 204

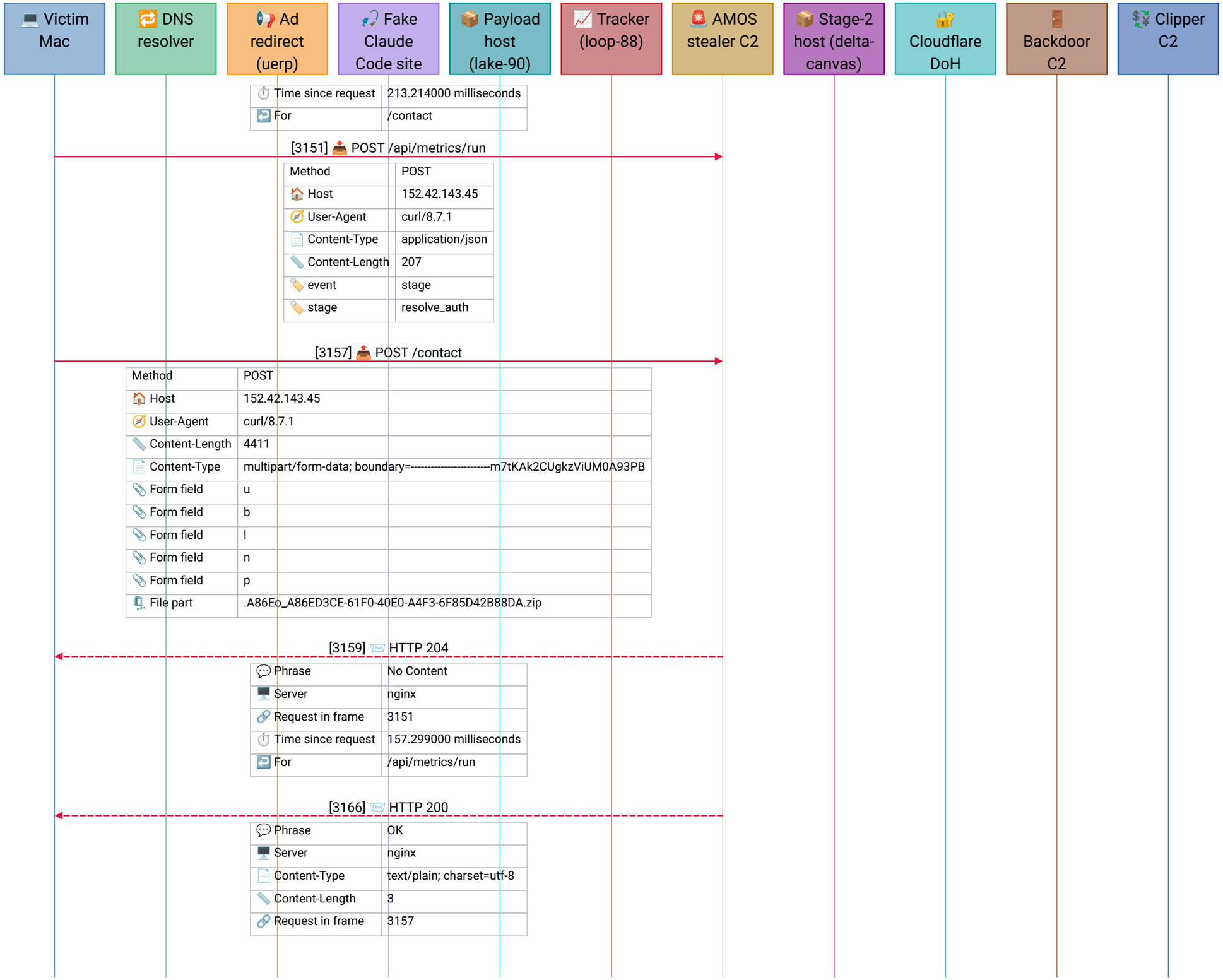
💬 Phrase	No Content
🖥️ Server	nginx
🔗 Request in frame	3007
⏱️ Time since request	148.281000 milliseconds
↩️ For	/api/metrics/run

[Frame 3027] `204` for the `passmgr` report (frame 3007).

[3036] 📨 HTTP 200

💬 Phrase	OK
🖥️ Server	nginx
📄 Content-Type	text/plain; charset=utf-8
📏 Content-Length	3
🔗 Request in frame	3015

[Frame 3036] The C2 server accepts the first upload with `200 OK` and a 3-byte body, `OK` and a newline.



[Frame 3151]

Phase 4: The stolen password

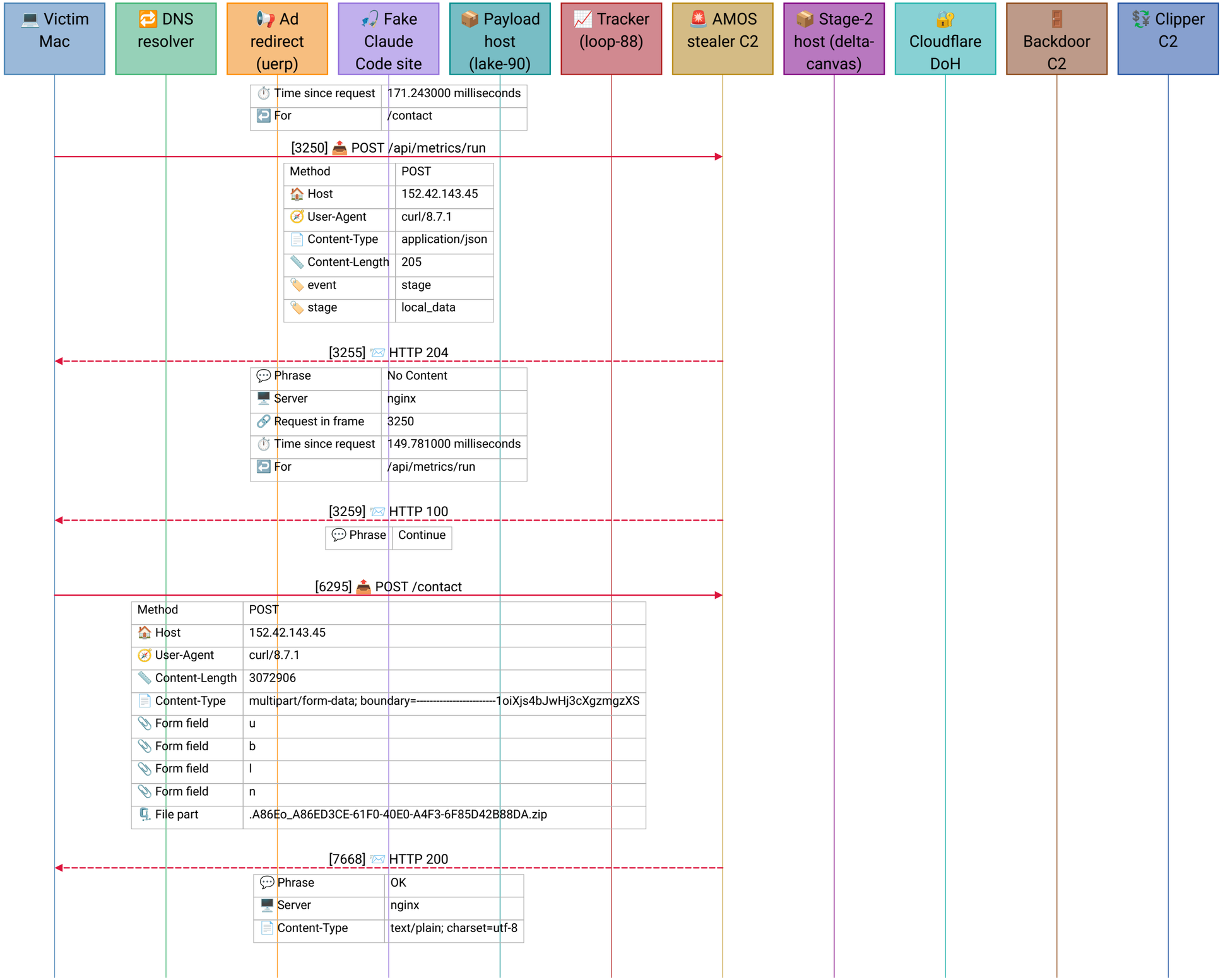
110 seconds of silence, then `stage: resolve_auth`. In the gap, the stealer showed the fake system dialog seen in Brad's screenshots, asking for the Mac's password. The pause is the time it took a person to read the dialog and type.

[Frame 3157] The password leaves the Mac. The second upload repeats the first, plus one new 9-byte file in the zip: `pwd`, the macOS login password typed into the fake dialog.

On a Mac, the login password also protects the user's keychain, the encrypted store of saved passwords. The next upload takes the keychain files.

[Frame 3159] `204` for the `resolve_auth` report (frame 3151).

[Frame 3166] `200 OK` for the second upload.



[Frame 3250]

Phase 5: The bulk upload

`stage: local_data`, 35 seconds later: the stealer has finished gathering files from the disk.

[Frame 3255] `204` for the `local_data` report.

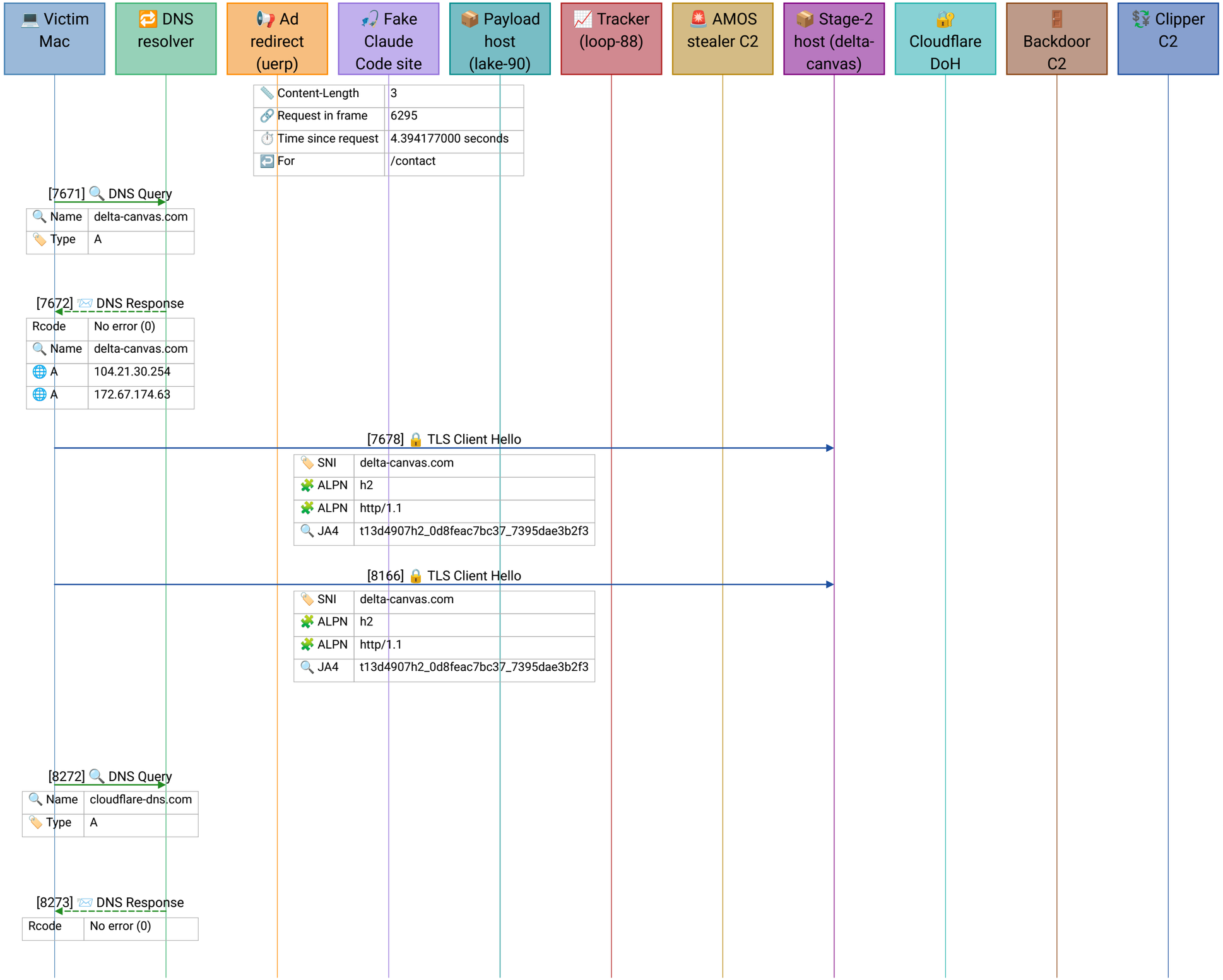
[Frame 3259] **`100 Continue.`** For a large body, `curl` sends the headers first with `Expect: 100-continue` (frame 3253) and waits for the server's go-ahead. The headers already declare the size: `Content-Length: 3072906.`

[Frame 6295] The bulk theft: 3 MB in 2.3 seconds. Wireshark completes this request after reassembling about 2,250 TCP segments. The zip, 3,072,183 bytes, adds the most damaging items:

- `keychain-2.db` with its WAL and shared-memory files and the `user.kb` keybag (about 2.6 MB): the keychain database. The password from frame 3157 is what the attackers need to try to decrypt it.
- Five screenshots from the desktop.
- A process list and a list of installed software, which show what security tools are running.
- Apple Notes' database files (empty on this test Mac).

All of it crosses the network unencrypted, which is why the full contents can be read from the capture.

[Frame 7668] `200 OK`, 4.4 seconds after the last byte. Brad's notes show that the stealer then deleted its staging folder and zip from `/tmp`, removing the evidence on disk. The network evidence remains.



[Frame 7671]

Phase 6: Staying on the Mac

Within a second of the upload, the Mac looks up a new domain, delta-canvas [.]com. The stealing is done; this phase sets up persistence.

[Frame 7672] Another Cloudflare-fronted domain, the fourth in this infection.

[Frame 7678] `curl` 's fingerprint again. This connection downloads about 0.54 MB on the wire, which fits the size of one of the two persistent programs Brad found, `AccountsHelper` (504,752 bytes). The name mimics Apple's `accountsd` service.

[Frame 8166] A second download of about 0.57 MB on the wire, fitting the other persistent program, `mdworker_shared` (535,440 bytes), named after a real macOS Spotlight process.

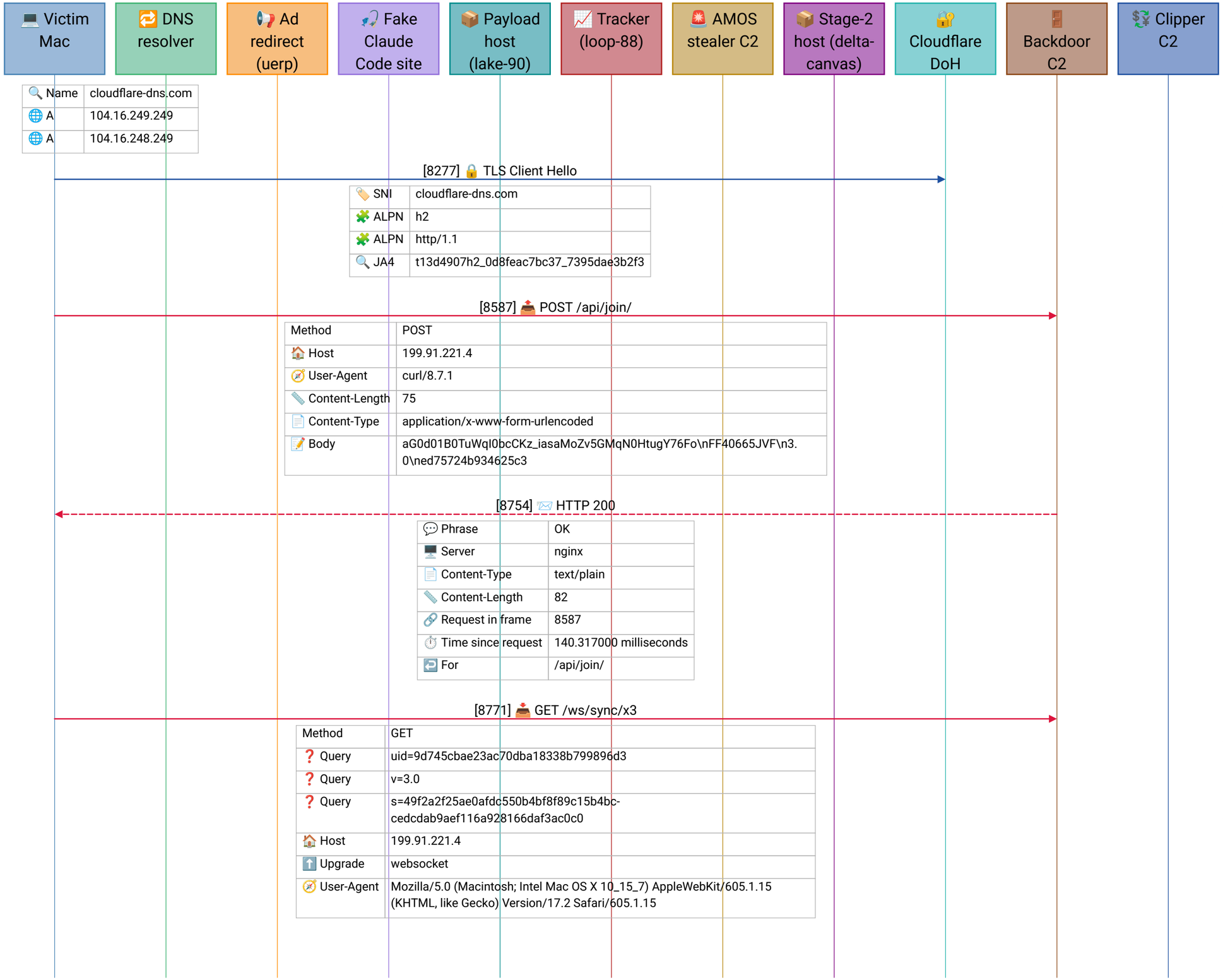
Brad's notes show both installed as `LaunchDaemons` (`/Library/LaunchDaemons/com.apple.accountsd.helper.plist` and `com.apple.metadata.mds.worker.plist)`, which macOS starts at every boot, normally as root. Whether each download feeds the backdoor or the clipper is inferred from the order and sizes, not visible in the encrypted traffic.

[Frame 8272]

Phase 7: A backdoor checks in

The Mac looks up cloudflare-dns [.]com, Cloudflare's legitimate DNS-over-HTTPS (DoH) service. Browsers can use DoH, but the next Client Hello carries `curl` 's fingerprint, not Safari's.

[Frame 8273] Cloudflare's DoH addresses. The name is legitimate, so it would never appear on a blocklist.



[Frame 8277] `curl` 's fingerprint, to cloudflare-dns [.]com. Within 0.3 seconds of this connection opening, the Mac contacts a new C2 address it never looked up through ordinary DNS. The likely explanation: the backdoor resolves its C2 server over encrypted DoH, hiding the lookup from the network's resolver and its logs.

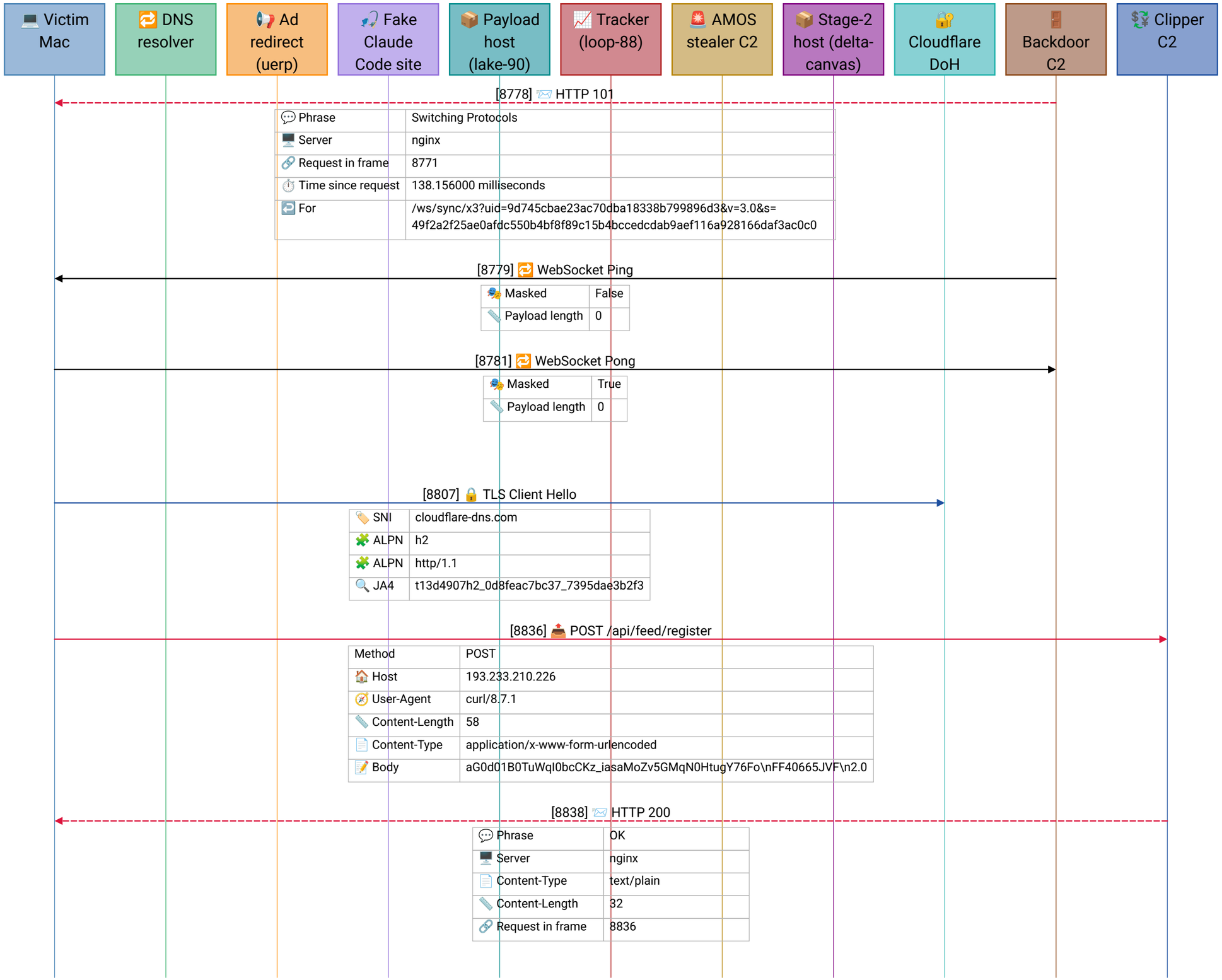
[Frame 8587] Backdoor registration. A POST `/api/join/` to 199.91.221 [.]4, again plain HTTP to a bare IP. The body is four lines: the same victim ID u the stealer used in frame 2962, a 10-character identifier `FF40665JVF`, a version `3.0`, and a 16-hex-digit value.

Reusing the stealer's victim ID ties the backdoor to the stealer infection on the operators' side.

[Frame 8754] The C2 server assigns the bot an ID, `nXRcuuI6xw26GDOLeZiW0w`, and a token on a second line. Watch for this ID: it returns 15 minutes later, in frame 27884.

[Frame 8771] A WebSocket with a borrowed identity. The backdoor asks to upgrade a new HTTP connection to a WebSocket at `/ws/sync/x3`, with a `uid`, a version, and a 64-hex-digit value s (likely a signature) in the query string.

Every request so far said `curl/8.7.1`. This one claims to be Safari 17.2, a version from December 2023, on a Mac that frame 2962 reported as running macOS 27. A copied browser `User-Agent` is a common disguise for non-browser traffic, and here the capture contradicts it.



[Frame 8778] `101 Switching Protocols`: the server accepts the upgrade. From now on the TCP connection carries WebSocket frames, a channel the server can push commands down at any time.

[Frame 8779] The server sends the first WebSocket `Ping`, 0.04 ms after the 101.

[Frame 8781] The backdoor answers with a `Pong`. It is masked, as RFC 6455 requires of every client-to-server frame.

The pair repeats every 25 seconds for the next 15 minutes: 37 pings in all, of which the diagram shows the first and the last. A perfectly regular heartbeat to a bare IP address is a strong beaconing signal, even when no commands are sent.

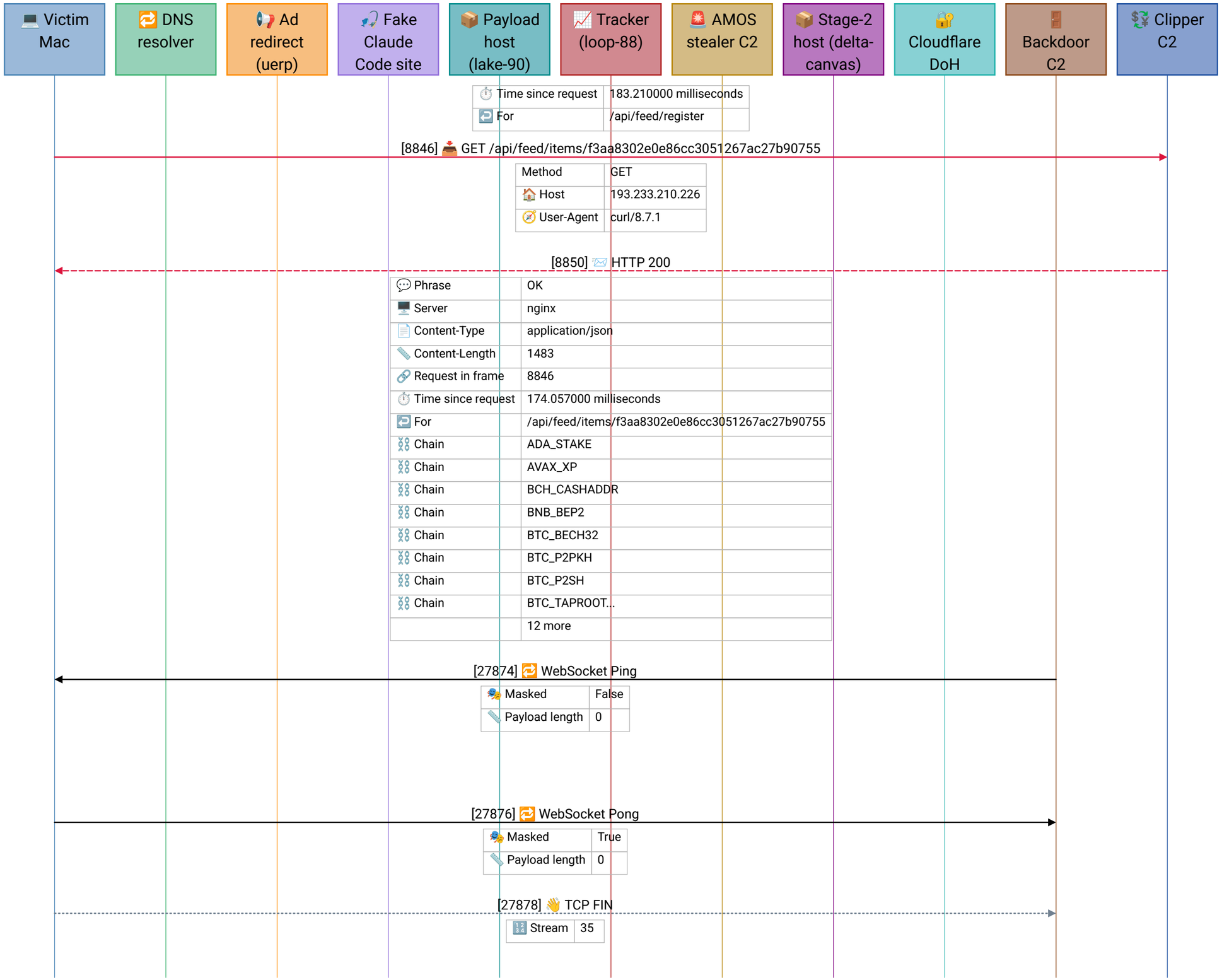
[Frame 8807]

Phase 8: A clipboard hijacker loads its address list

A second DoH connection, about half a second before the Mac contacts a C2 address it never resolved through ordinary DNS.

[Frame 8836] A second implant registers with a third C2 server, 193.233.210 [.]226: `POST /api/feed/register` with the same victim ID and identifier as frame 8587, and version `2.0`.

[Frame 8838] The server returns a 32-hex-digit feed ID, `f3aa8302...`, which the next request uses.



[Frame 8846] The implant fetches its feed: `GET /api/feed/items/f3aa8302....`

[Frame 8850] The clipper's rule list. The JSON reply (`version: 5`) holds 19 rules, each pairing an address format with an attacker-owned address. They cover 15 blockchains: Bitcoin has four rules, one per address format (legacy `1...`, script `3...`, SegWit `bc1q...`, Taproot `bc1p...`), Litecoin two, and Ethereum, Solana, Dogecoin, XRP, Tron, Cardano, Avalanche, and six other chains one each.

This is a **clipper:** malware that watches the clipboard and, when the user copies a wallet address, silently swaps in the attacker's address of the same format. The victim pastes it into a payment and sends funds to the attacker. Each format gets its own rule because a swapped address must match the format of the one the user copied.

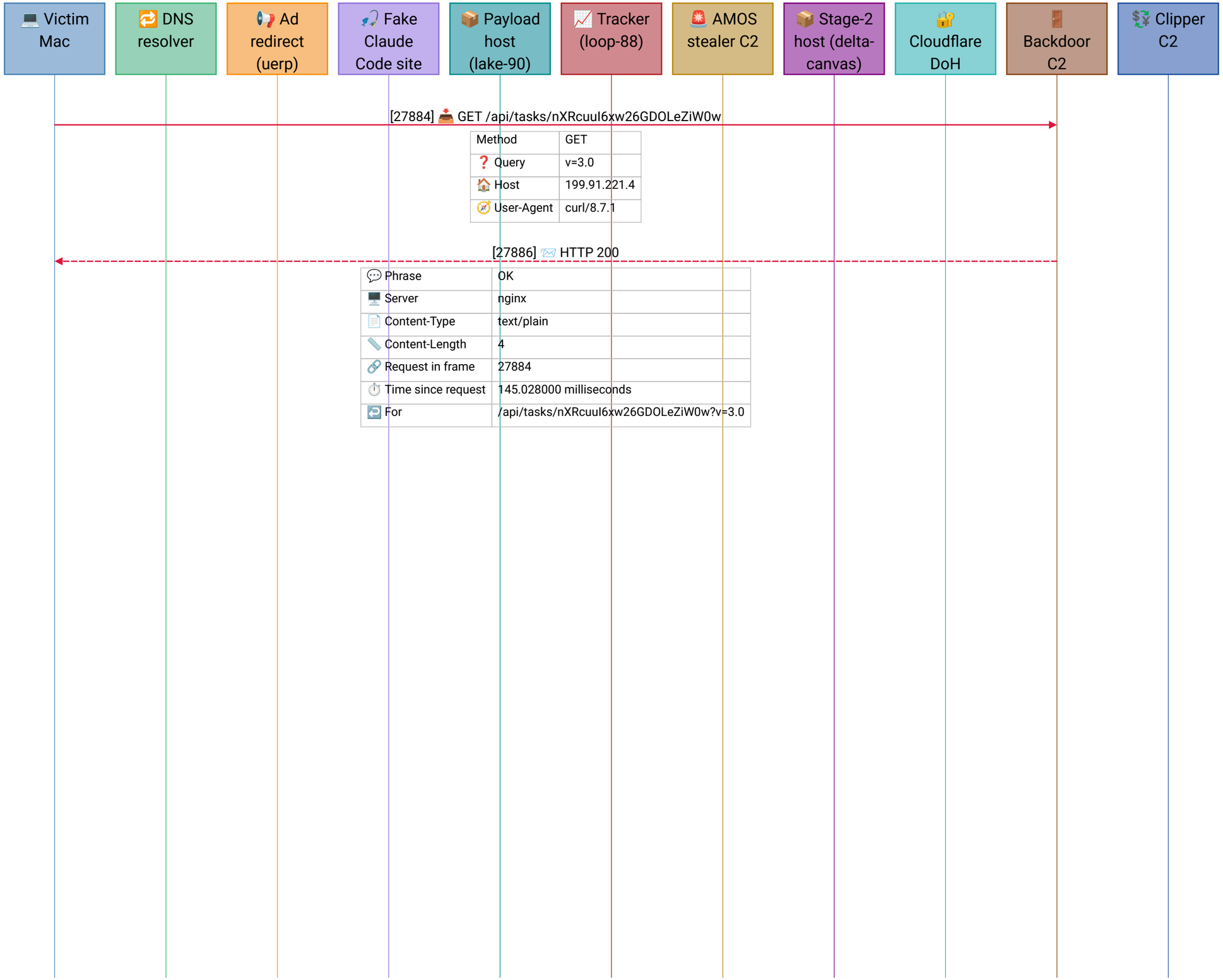
[Frame 27874]

Phase 9: Fifteen minutes later

The 37th `Ping`, at 22:44:42, exactly 900 seconds (36 × 25 s) after the first. The heartbeat never missed. Meanwhile, the capture between these frames is Brad's ordinary browsing (weather, news, social sites), left out of the diagram.

[Frame 27876] The 37th `Pong`.

[Frame 27878] The Mac closes the WebSocket itself, less than a second after its last pong: a plain TCP `FIN`, with no WebSocket close message. The socket was open for 900.9 seconds, almost exactly 15 minutes, which points to a timer



rather than a failure.

[Frame 27884] 35 ms after the FIN, before the server has acknowledged it, the Mac opens a new connection to the same server. Over it, the backdoor polls over plain HTTP: `GET /api/tasks/nXRcuuI6xw26GDOLeZiW0w`, using the bot ID the C2 server assigned in frame 8754. The `User-Agent` is `curl` again.

[Frame 27886] The server's whole reply is one word, `pong`. With frame 27878, this looks like a 15-minute cycle: hold the WebSocket, drop it, check for queued tasks over HTTP, and probably reconnect. The pong could mean "no task", or it could simply answer a keepalive; one cycle cannot tell them apart, and the capture ends 17 seconds later, before any reconnect.

When the capture ends, the Mac is fully compromised: its passwords, keychain, and files gone; a backdoor waiting for orders; and a clipper ready to redirect any cryptocurrency payment.