SecuritySpy
Live Video Stream

Protocol specification for the `video` HTTP multipart stream
Motion JPEG, WebP, H.264 and H.265

Applies to: SecuritySpy 6 and later

Endpoint: `GET /video`

Audience: Developers writing third-party clients that consume live video from SecuritySpy

Document revision: 1.0

Contents

1 Overview

SecuritySpy's built-in web server exposes a live video stream at the `video` endpoint. The transport is **HTTP `multipart/x-mixed-replace`** — the same "server push" framing used by the ubiquitous MJPEG streams produced by IP cameras. A client issues a single `GET` request, and the server replies with an HTTP response whose body never ends: a sequence of MIME parts, each carrying one complete video frame, streamed for as long as the connection stays open.

Where SecuritySpy departs from a conventional MJPEG server is in what those parts may contain. Rather than being restricted to JPEG, each part can carry:

- **JPEG** — a complete, self-contained image per part. Any MJPEG client can consume this.
- **WebP** — a complete, self-contained image per part; roughly half the bytes of an equivalent JPEG.
- **H.264** — one access unit (one coded frame) per part, as length-prefixed NAL units.
- **H.265 (HEVC)** — the same, using HEVC NAL units.

The inter-frame codecs are the reason this document exists. Because H.264 and H.265 frames are not self-contained, three extra pieces of information have to travel alongside the payload, and SecuritySpy carries them in custom per-part MIME headers:

- `SS-Config` — the parameter sets (SPS/PPS, plus VPS for H.265) needed to initialise a decoder.
- `SS-Delta` — whether this part is a key frame or a delta (inter-coded) frame.
- `SS-Timestamp` — the presentation time of the frame.

Everything else — the boundary framing, `Content-Length`, `Content-Type` — is standard, so a client that already speaks MJPEG needs only modest additions to handle H.264 and H.265.

> **WHY BOTHER WITH H.264/H.265?**
> For a typical 1080p camera, an H.264 stream is an order of magnitude smaller than the equivalent JPEG stream at the same frame rate. Just as importantly, when the requested stream is close enough to what the camera is already producing, SecuritySpy *passes the camera's compressed data straight through* without decoding or re-encoding it — so the stream costs the Mac almost no CPU, and is bit-for-bit the quality the camera produced. See section 10.

2 Endpoint and request

2.1 URL

The stream is requested with a plain HTTP `GET`:

```
GET /video?cameraNum=0&vcodec=h264 HTTP/1.1
Host: 192.168.1.10:8000
Authorization: Basic dXNlcjpwYXNz
```

The server listens on HTTP and HTTPS, by default on ports **8000** and **8001** respectively, though both are user-configurable. HTTPS is strongly recommended for any stream that leaves the local network, since credentials are otherwise sent in the clear.

2.2 Connection lifetime

The response has **no** `Content-Length`. The stream ends only when one side closes the TCP connection. There is no terminating boundary delimiter — a client must not wait for one.

To stop a stream, simply close the socket. SecuritySpy detects the closure and tears down the encoder for that connection.

3 Authentication

Unless the web server has been configured to allow anonymous access, the request must be authenticated. The account used must hold the *live video* permission for the requested camera, or the request is rejected with `401` (or `403`, see section 13).

SecuritySpy accepts credentials in two ways, checked in this order:

1. The `auth` query parameter

The value is the Base64 encoding of `username:password`, percent-encoded so that any `=` padding characters survive the URL. This is convenient for embedding a stream in an `<img>` tag or passing a URL to a tool that cannot set headers.

```
// user "admin", password "secret"  ->  "admin:secret"  ->  base64 "YWRtaW46c2VjcmV0"
GET /video?cameraNum=0&auth=YWRtaW46c2VjcmV0
```

2. HTTP Basic authentication

A standard `Authorization: Basic <base64>` request header. This is the recommended mechanism for third-party applications.

> **PER-ACCOUNT LIMITS**
>
> An account may be configured with a maximum frame rate, a maximum resolution, a maximum stream duration, and a permitted time-of-day window. The frame rate and resolution limits are applied silently on top of whatever the client requests — the stream you receive may be slower or smaller than the one you asked for, and the `SS-Width` / `SS-Height` response headers tell you what you actually got. The duration limit and the time window are enforced by *closing the connection*, with no warning in the multipart stream.

> **BRUTE-FORCE PROTECTION**
>
> Ten failed logins from one IP address cause subsequent connections from that address to be dropped at accept time. A client that retries a bad password in a tight loop will lock itself out.

4 Query parameters

All parameters are optional except `cameraNum`. Unrecognised parameters are ignored.

Parameter	Type	Meaning
`cameraNum`	integer *required*	Zero-based camera number. Must identify a camera that exists, is enabled and is currently online, otherwise the request fails with `404`. Camera numbers can be discovered via the `systemInfo` request.
`vcodec`	string	Video codec for the stream. One of: • `h264` or `avc1` — H.264 • `h265` or `hvc1` — H.265 / HEVC • `h26x` — "either"; the server picks H.265 if it can pass the camera's H.265 through untouched, otherwise H.264 • `webp` — WebP still images **If omitted, the stream is JPEG.** Any unrecognised value also yields JPEG.
`fps`	float	Requested frame rate. Only honoured if it is *lower* than the camera's current frame rate; you can never ask for more frames than the camera is producing. Ignored when H.264/H.265 passthrough is in effect (see 10.2).
`width`	integer	Requested output width in pixels. If `height` is omitted, the height is derived from the camera's aspect ratio. Clamped to the camera's native width; a value within 3 pixels of native, or out of range, means "native size".
`height`	integer	Requested output height in pixels. If `width` is omitted, the width is derived from the aspect ratio. Clamped as above.
`sizeFraction`	integer	Convenience alternative to `width` / `height`: divides the camera's dimensions by this integer. `sizeFraction=2` gives a half-size stream, `=4` a quarter-size stream. Values greater than 1 take precedence over `width` and `height`.
`quality`	integer 1–100	Encoder quality. Applies only when SecuritySpy is actually encoding — it has no effect on a passed-through H.264/H.265 stream, which is always at the camera's own quality. Defaults to the global preference.
`acodec`	string	Interleave the camera's live audio into the same stream. `ulaw` (G.711 μ-law) or `aac`. Requires the camera to have audio and the account to hold the *get audio* permission. Omit for video only. See section 11.
`vpause`	0 or 1	`vpause=1` opens the connection but starts with video paused (no frames sent). Chiefly useful for audio-only use of this endpoint.
`apause`	0 or 1	`apause=1` starts with audio paused. Only meaningful together with `acodec`.
`auth`	string	Credentials in the URL — see section 3.

4.1 Examples

```
# Classic MJPEG, full size, full frame rate
/video?cameraNum=0

# MJPEG, quarter size, 5 fps, moderate quality
/video?cameraNum=0&sizeFraction=2&fps=5&quality=60

# H.264 specifically — passed through only if the camera itself produces H.264,
# otherwise SecuritySpy has to re-encode
/video?cameraNum=3&vcodec=h264

# Best available inter-frame codec, letting the server choose — the highest
# chance of a zero-CPU passthrough of the camera's own stream
/video?cameraNum=3&vcodec=h26x

# H.265 with interleaved G.711 audio
/video?cameraNum=3&vcodec=h265&acodec=ulaw
```

5 The HTTP response

On success the server replies `200 OK`. A representative response header:

```
HTTP/1.1 200 OK
Server: BBVS/6.0.0/0FE43C21-8C6B-4E5A-9E4B-1A2B3C4D5E6F
Cache-Control: max-age=0, must-revalidate
Pragma: no-cache
SS-UUID: 0FE43C21-8C6B-4E5A-9E4B-1A2B3C4D5E6F
Keep-Alive: timeout=20, max=100
Connection: Keep-Alive
Content-Type: multipart/x-mixed-replace;boundary=ssBoundary8345
SS-PTZ: 15
SS-Width: 1920
SS-Height: 1080
Content-Security-Policy: frame-ancestors 'self'

--ssBoundary8345
...
```

Header	Meaning
`Content-Type`	Always `multipart/x-mixed-replace;boundary=ssBoundary8345`. The boundary token is a fixed literal and does not vary between servers, cameras or sessions.
`SS-UUID`	Unique identifier of this SecuritySpy installation.
`Server`	`BBVS/<version>/<uuid>`. Useful for feature detection.
`SS-Width`, `SS-Height`	The dimensions of the video frames carried in this stream.

Header	Meaning
`SS-PTZ`	Bitmask of the pan/tilt/zoom capabilities of this camera. Informational; not needed to decode the stream.
`Content-Length`	**Absent.** The body is unbounded.

6 Stream structure

The body is a sequence of MIME parts. Each part is, byte for byte:

```
--ssBoundary8345<CR><LF>
Content-Length: <n><CR><LF>
Content-Type: <mime><CR><LF>
<zero or more additional SS-* headers, each ending <CR><LF>>
<CR><LF>
<exactly n bytes of payload>
<CR><LF>
```

Points to note:

- The boundary delimiter is the literal ASCII `--ssBoundary8345`. There is **no** leading `<CR><LF>` before the first one — it follows the blank line that terminates the HTTP response header.
- `Content-Length` is always present and always exact. **Use it.** Do not scan for the boundary string inside binary payloads: H.264, H.265 and WebP data can and will contain the byte sequence `--ssBoundary8345` by coincidence.
- Each payload is followed by a single `<CR><LF>`, then the next boundary line.
- There is no closing `--ssBoundary8345--` delimiter. The stream ends when the connection does.
- Header order within a part is stable but a client should parse headers by name, not by position.

6.1 Per-part headers

Header	When present	Meaning
`Content-Length`	Always	Exact byte count of the payload that follows the blank line.
`Content-Type`	Always	One of `image/jpeg`, `image/webp`, `image/avc1` (H.264), `image/hvc1` (H.265) — or, if audio was requested, `audio/basic` / `audio/aac`. **This is the field that tells a part apart from its neighbours; branch on it.**
`SS-Timestamp`	Every video part	Presentation timestamp of this frame. See section 9.
`SS-Delta`	H.264 / H.265 video parts	`0` = this access unit is a key frame (IDR); `1` = it is a delta (inter-coded) frame that depends on earlier frames. Never sent for JPEG or WebP, where every part is self-contained.

Header	When present	Meaning
`SS-Config`	H.264 / H.265, first key frame and whenever the encoder configuration changes	The codec parameter sets. See section 8. A decoder cannot be initialised without these.
`SS-X`	Occasionally	Base64 camera-status metadata; appears on the first frame and thereafter only when the status changes. See 12.1.
`SS-Attention`	Every video part	Base64 bitmap of which cameras currently need attention. See 12.2.

`SS-Audio-Sample-Rate`
`SS-Audio-Channel-Count`
`SS-Audio-Magic-Cookie`
	First AAC audio part only	AAC stream configuration. See section 11.

> **Why `image/avc1`?**
>
> The MIME type is formed mechanically from the codec's four-character code (`avc1` for H.264, `hvc1` for H.265), so H.264 parts are labelled `image/avc1` and H.265 parts `image/hvc1`. These are not registered MIME types and are specific to SecuritySpy; match on them literally.

6.2 A complete JPEG part

```
--ssBoundary8345
Content-Length: 47213
Content-Type: image/jpeg
SS-Timestamp: 0
SS-X: AAAAAQEBAAAAAAAAAAA=
SS-Attention: AA==

<47213 bytes: FF D8 FF E0 ... FF D9>
```

6.3 A complete H.264 key-frame part

```
--ssBoundary8345
Content-Length: 68402
Content-Type: image/avc1
SS-Config: profile-level-id=4D0029;sprop-parameter-
sets=Z01AKZpmA8ARPy4C3AQEBQAAAwPoAAHUwHjBjLA=,aO48gA==
SS-Delta: 0
SS-Timestamp: 0
SS-X: AAAAAQEBAAAAAAAAAAA=
SS-Attention: AA==

<68402 bytes: 4-byte-length-prefixed NAL units>
```

…followed by delta frames, which are much smaller and carry no `SS-Config`:

```
--ssBoundary8345
Content-Length: 3140
Content-Type: image/avc1
SS-Delta: 1
SS-Timestamp: 40
SS-Attention: AA==

<3140 bytes>
```

7 Video payloads by codec

7.1 JPEG — `Content-Type: image/jpeg`

The payload is a complete, standalone JFIF/JPEG file, beginning `FF D8` and ending `FF D9`. Nothing else is required to display it. This is what you get when `vcodec` is omitted, and it is what makes the stream compatible with every existing MJPEG client, including a bare `<img src="...">` tag in a browser.

7.2 WebP — `Content-Type: image/webp`

The payload is a complete standalone WebP file (a RIFF container beginning `RIFF....WEBP`). Like JPEG, every part is independently decodable; unlike JPEG it is not supported by legacy MJPEG clients. It typically halves the bandwidth of a JPEG stream at comparable visual quality, but it can use significant CPU on the server to produce, as it is more complex than JPEG, and its production cannot be hardware-accelerated. Request it with `vcodec=webp`.

7.3 H.264 — `Content-Type: image/avc1`

Each part carries **exactly one access unit** — one coded video frame, comprising one or more NAL units.

The NAL units are in **length-prefixed (AVCC / "MP4") form**, not Annex B start-code form:

```
┌──────────────────┬──────────────────────┬──────────────────┬──────────────────┐
│ length (4 B)     │ NAL unit (length B)  │ length (4 B)     │ NAL unit ...     │
│ big-endian       │ starts with NAL hdr  │ big-endian       │                  │
└──────────────────┴──────────────────────┴──────────────────┴──────────────────┘
```

- The length prefix is **4 bytes, big-endian (network byte order)**, and gives the size of the NAL unit that follows, *excluding* the prefix itself.
- There are **no** `00 00 00 01` start codes and **no** emulation-prevention removal to do — the NAL payloads are exactly as the encoder or camera produced them.
- NAL units repeat until `Content-Length` bytes have been consumed. Do not assume one NAL unit per part.
- The NAL unit type is the low 5 bits of the first byte of each NAL unit: `nal[0] & 0x1F` (type 5 = IDR slice, type 1 = non-IDR slice, 7 = SPS, 8 = PPS).

- SPS and PPS are **not** normally included in the payload. They are delivered out of band in the `SS-Config` header — see section 8.

To feed a decoder that wants Annex B (FFmpeg's `h264` parser, most hardware decoders on Linux), replace each 4-byte length prefix with the 4-byte start code `00 00 00 01`, and prepend the SPS and PPS (each with its own start code) before the first key frame. To feed VideoToolbox on Apple platforms, hand the length-prefixed data straight to a `CMBlockBuffer` and build a `CMVideoFormatDescription` from the parameter sets — no conversion is needed at all.

7.4 H.265 / HEVC — `Content-Type: image/hvc1`

Byte-for-byte identical framing to H.264: one access unit per part, NAL units carried with **4-byte big-endian length prefixes**, no start codes.

The differences are in the NAL units themselves:

- The HEVC NAL header is **2 bytes**, not 1. The NAL unit type is the middle 6 bits of the first byte: `(nal[0] >> 1) & 0x3F`.
- Relevant types: 32 = VPS, 33 = SPS, 34 = PPS; types 16–21 are IRAP (key) frames, of which 19/20 are IDR.
- There are **three** parameter sets to configure a decoder — VPS, SPS *and* PPS — all carried in `SS-Config`.

> **ONE ACCESS UNIT PER PART — ALWAYS**
> SecuritySpy never splits a frame across parts and never packs two frames into one part. One MIME part is exactly one displayable frame. This is a stronger guarantee than RTP gives you, and it means a client needs no access-unit delimiter logic and no reassembly buffer.

8 H.264 / H.265 parameter sets: the `SS-Config` header

A decoder cannot start without the codec's parameter sets. SecuritySpy sends them in the `SS-Config` per-part header, on the first key frame of the stream and again on any subsequent key frame where the camera's configuration has changed (a resolution change, for example). The format deliberately mirrors the SDP `fmtp` line used by RTSP, so the parsing code you may already have will work unmodified.

8.1 H.264

```
SS-Config: profile-level-id=<PPIILL>;sprop-parameter-sets=<SPS-base64>,<PPS-base64>
```

`profile-level-id`	Six hexadecimal digits: profile_idc, profile_iop (constraint flags) and level_idc, one byte each. E.g. `4D0029` = Main profile, level 4.1.
`sprop-parameter-sets`	Two Base64 blobs separated by a comma: the SPS, then the PPS. Each blob decodes to a **raw NAL unit** — the NAL header byte followed by the RBSP. There is no start code and no length prefix.

Worked example:

```
SS-Config: profile-level-id=4D0029;sprop-parameter-
sets=Z01AKZpmA8ARPy4C3AQEBQAAAwPoAAHUwHjBjLA=,aO48gA==

base64("Z01AKZpm...")  ->  67 4D 40 29 9A 66 03 C0 ...   // 0x67 & 0x1F = 7  -> SPS
base64("aO48gA==")     ->  68 EE 3C 80                   // 0x68 & 0x1F = 8  -> PPS
```

8.2 H.265

```
SS-Config: profile-level-id=<space>.<profile>.<level>;sprop-sps=<b64>;sprop-pps=<b64>;sprop-
vps=<b64>
```

`profile-level-id`	Three decimal values separated by dots: profile_space, profile_idc, level_idc. E.g. `0.1.120` = Main profile, level 4.0.
`sprop-sps`	Base64 of the raw SPS NAL unit.
`sprop-pps`	Base64 of the raw PPS NAL unit.
`sprop-vps`	Base64 of the raw VPS NAL unit.

8.3 Rules for clients

- Do not attempt to decode any frame before you have seen an `SS-Config` header. It is guaranteed to arrive on the first frame you receive.
- When `SS-Config` appears again mid-stream, the stream format has changed. Tear down the decoder, rebuild it from the new parameter sets, and resume. The part carrying a new `SS-Config` is always a key frame (`SS-Delta: 0`).
- The first video part of the stream is always a key frame. If you reconnect, you will always resynchronise cleanly — the server will not start you mid-GOP.

9 Timestamps

```
SS-Timestamp: 40
```

Units	Ticks of a **600 Hz** clock. One tick = 1/600 s ≈ 1.667 ms. (600 is chosen because it divides evenly by 24, 25, 30, 50 and 60.)
Epoch	**Stream-relative.** The first frame of the stream is timestamp `0`; every subsequent value is the offset from that first frame. It is not a wall-clock time and cannot be converted into one.
Type	Signed 64-bit decimal integer, monotonically increasing.
Conversion	`seconds = SS-Timestamp / 600.0`

Timestamps come from the camera's own clock, so the interval between consecutive frames reflects the real capture cadence — including jitter and any frames the camera or the network dropped. A client rendering the stream should schedule presentation from these timestamps rather than assuming a fixed frame interval.

10 Passthrough versus re-encoding

This section explains why the stream you get may not be exactly the stream you asked for — and why that is usually what you want.

10.1 The two paths

When you request H.264 or H.265, SecuritySpy chooses one of two strategies:

Passthrough (preferred)

The compressed frames arriving from the camera are forwarded to you untouched. No decoding, no scaling, no re-encoding. This costs essentially no CPU, adds no latency, and delivers exactly the quality the camera produced — which is the best quality obtainable. It is what makes it practical to serve many simultaneous H.264 clients from one Mac.

Re-encoding (fallback)

SecuritySpy decodes the camera's video, applies any requested scaling and image adjustments, and re-encodes it with VideoToolbox at the requested size, frame rate and quality. This costs CPU/GPU per connection, and quality is bounded by the encode. A re-encoded H.264/H.265 stream is given a **key-frame interval of roughly three seconds** (`ceil(fps × 3)` frames), so a client that joins or resynchronises will never wait long for a key frame.

10.2 When passthrough is used

Passthrough is chosen only when *every* one of the following holds:

- The camera is a **network camera** (not a local USB/Blackmagic device, and not a "push" source).
- The camera is natively producing **the codec you asked for**. Ask for H.265 from an H.264 camera and you will get a re-encode; ask for `vcodec=h26x` and SecuritySpy will pick whichever avoids that.
- The camera's **key-frame interval is not excessive** — no longer than 5 seconds' worth of frames. A camera configured with a 30-second GOP would leave a new client staring at a blank screen, so SecuritySpy re-encodes instead.
- The camera has **no privacy mask** and **no image adjustments** (brightness, contrast, rotation, de-fisheye…) configured — these can only be applied by decoding.
- Your **requested frame rate is close to native**: within 4 fps of the camera's rate, or unspecified. Asking for a much lower frame rate forces a re-encode.
- Your **requested size is close to native**: at least half the camera's width and half its height. Asking for a small thumbnail forces a re-encode.

- The connection is on the **local network**, *or* the "video passthrough" web setting is enabled for remote connections.

Consequences a client must be prepared for:

- **`fps`, `width`, `height`, `quality` and `sizeFraction` are requests, not commands.** Under passthrough, the frame rate and dimensions are the camera's, and `quality` is meaningless.
- Under passthrough, **no frame-rate limiting is applied** — every frame the camera sends, you get.
- Take the true dimensions from the SPS, not from `SS-Width` / `SS-Height`.

> **THE RECOMMENDED WAY TO ASK**
>
> For a full-quality, low-CPU live view, request `vcodec=h26x` and specify *nothing else* — no size, no fps, no quality. This lets SecuritySpy hand you the camera's own stream verbatim, in whichever of the two codecs the camera is producing. Read the resulting `Content-Type` (`image/avc1` or `image/hvc1`) to learn which one you got.

10.3 JPEG

For JPEG there is no passthrough concept as such: if you request the camera's native size, SecuritySpy hands you the JPEG it has already produced for its own use, shared between all connections that want it — cheap. If you request any other size, a per-connection JPEG encoder is created. JPEG encoding is deliberately done in software rather than on the hardware encoder, so that large numbers of simultaneous JPEG streams cannot starve the hardware decoders.

11 Optional audio in the same stream

Adding `acodec` to the request interleaves the camera's live audio into the same multipart stream, as additional parts distinguished by their `Content-Type`. Audio parts and video parts are interleaved in capture order; a client that does not want audio simply omits the parameter, and a client that wants only some of it can skip any part whose `Content-Type` begins with `audio/`.

Audio requires that the camera has audio enabled and that the account holds the *get audio* permission; otherwise the parameter is silently ignored and you receive a video-only stream.

Two structural points differ from video parts:

- **Audio parts carry no `SS-Timestamp`.** Timing is implicit: the samples are contiguous, so presentation time follows from the running sample count and the sample rate.
- **No audio part is ever sent before the first video part**, because the HTTP response header itself is constructed with that first video frame.

11.1 `acodec=ulaw` — G.711 μ-law

```
--ssBoundary8345
Content-Length: 1024
Content-Type: audio/basic

<1024 bytes of 8-bit G.711 μ-law samples>
```

Always **8000 Hz, mono, 8 bits per sample**, regardless of the camera's native audio format — SecuritySpy resamples and converts as necessary. This is the simplest option and the one to reach for first: the payload is raw μ-law samples with no container and no header, playable by feeding it directly to any G.711 decoder.

11.2 `acodec=aac` — AAC

AAC parts carry the source's native sample rate and channel count. Because an AAC decoder needs configuration data, the **first** AAC part carries three extra headers:

```
--ssBoundary8345
Content-Length: 384
Content-Type: audio/aac
SS-Audio-Channel-Count: 1
SS-Audio-Sample-Rate: 16000
SS-Audio-Magic-Cookie: A4AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...

<384 bytes of AAC data>
```

`SS-Audio-Sample-Rate`	Sample rate in Hz.
`SS-Audio-Channel-Count`	1 (mono) or 2 (stereo).
`SS-Audio-Magic-Cookie`	Base64 of the MPEG-4 `esds` elementary stream descriptor, which contains the AudioSpecificConfig needed to initialise the decoder.

Subsequent AAC parts carry the payload only. If your client cannot conveniently parse an `esds`, use `acodec=ulaw` instead — it needs no configuration at all.

12 Metadata headers

These headers carry SecuritySpy application state alongside the video. They are entirely optional to consume — a client interested only in pictures may ignore them.

12.1 `SS-X` — camera status

Base64 of a packed 16-byte big-endian structure. It is sent on the first frame and thereafter only when the value changes, so a client must cache the last value it saw.

```
offset  size  field              meaning
------  ----  -----------------  ------------------------------------------------
   0     1    cStatus            Continuous-capture: 0 = off, 1 = armed, 2 = active
   1     1    mStatus            Motion-capture:     0 = off, 1 = armed, 2 = active
   2     1    aStatus            Actions:            0 = off, 1 = armed, 2 = active
   3     1    autoPatrolStatus   0 = unavailable, 1 = off, 2 = on
   4     1    cOverride          Continuous-capture schedule override type
   5     1    mOverride          Motion-capture schedule override type
   6     1    aOverride          Actions schedule override type
   7     1    (reserved)
   8     2    cSchedule          Continuous-capture schedule id  (big-endian uint16)
  10     2    mSchedule          Motion-capture schedule id      (big-endian uint16)
  12     2    aSchedule          Actions schedule id             (big-endian uint16)
  14     2    (reserved)
```

The structure may be extended in future versions; a client should tolerate a longer decoded value and read only the fields it knows.

12.2 `SS-Attention` — cameras needing attention

Base64 of a bitmap, one bit per camera, least-significant bit first within each byte: bit *n* of the decoded bytes is set if camera *n* currently requires attention. Trailing zero bytes are stripped, so the decoded value is often very short (`AA==` decodes to a single zero byte, meaning "no camera needs attention"). Sent on every video part.

12.3 `SS-PTZ` — pan/tilt/zoom capabilities

A decimal bitmask of the camera's PTZ features, sent once in the HTTP response header. Present so that a viewer can decide which on-screen controls to offer; it plays no part in decoding.

13 Errors and failure modes

13.1 Status codes

Code	Cause
`200 OK`	Stream is starting. Note that this header is not sent until the first frame is ready.
`401 Unauthorized`	Missing or bad credentials, or the account lacks live-video permission for this camera. Carries a `WWW-Authenticate: Basic` challenge.
`403 Forbidden`	Returned instead of `401` for connections where a browser login prompt would be unhelpful. Treat identically to `401`.
`404 Not Found`	No `cameraNum` given, camera number out of range, camera does not exist, camera is disabled, or camera is offline.

Code	Cause
`500 Internal Server Error`	The stream could not be set up — most commonly, the video encoder could not be started. The response body is a short plain-text description including an internal error code.

Error responses have a normal `Content-Length` and a plain-text body; they are not multipart.

13.2 Mid-stream failures

Once the stream is running there is no in-band error channel. If the encoder fails repeatedly, the camera goes offline, or the client cannot keep up, **SecuritySpy simply closes the connection.** A client should treat an unexpected EOF as a transient error and reconnect, ideally with a backoff.

13.3 Flow control and GOP dropping (H.264/H.265 passthrough)

If a passthrough client cannot read as fast as the camera produces data, the server's send buffer grows. When it exceeds twice the size of the incoming frame, SecuritySpy **drops the remainder of the current group of pictures** and waits for the next key frame before resuming. From the client's point of view this appears as a gap in `SS-Timestamp` followed by a fresh `SS-Delta: 0` frame.

A client must therefore be able to cope with discontinuities: never assume the next delta frame follows the previous one. Discard delta frames that arrive before you have seen your first key frame.

13.4 Variable frame rate

If the "variable frame rate" web setting is enabled on the server, JPEG and re-encoded streams may drop to a much lower frame rate for a camera with no current motion, rising again when motion occurs. This is invisible to the client apart from the frame timing, and is another reason to drive presentation from `SS-Timestamp`.

13.5 Server limits

Connections	Up to 1024 simultaneous web-server connections in total. There is no per-camera limit on the number of streams.
Idle timeout	A connection that makes no progress for 90 seconds is closed.
Frame rate	Never exceeds the camera's current frame rate; may be further limited by the account's maximum.
Resolution	Never exceeds the camera's native size; may be further limited by the account's maximum (expressed in megapixels).
Duration	If the account has a stream time limit, the connection is closed when it is reached.