E05. Payload capability re-examination

Ilja Krasnjanski
24 October 2026 · Edited evidence report
Historical evidence through 24 July 2026; later engineering checks identified separately.

This report synthesizes the cited case records. Exhibits distinguish retained artifact excerpts from secondary analytical records and claims whose original inputs are unavailable. Excerpts may be redacted or reformatted; extraction methods and limitations are stated locally. Fingerprints identify inputs, not the truth of their claims. Bracketed S labels identify the analysis records listed at the end.

Scope and method

The 24 July re-examination used static text searches, binary strings and archive listings across eighteen decrypted or text samples, a Go implant and a Windows agent bundle. No sample was executed. The analysis sought password-manager collection, keylogging indicators, shortener references and capture components. Conclusions here apply to the examined material and search methods. [S1]

Password-manager files

Two large JavaScript payloads include `.kdbx` in a private-key extension set used by collection predicates. This supports selection of encrypted KeePass databases through a generic file sweep. It does not establish a KeePass-specific parser, master-password recovery or successful database unlocking. Chance matches inside encoded dropper content were excluded. Negative searches for other password-manager names do not prove those products are unaffected. [S1]

Capture implementation

The Go binary contains capture-related symbols including `CaptureAndSend`, `Loop`, `encodeBlocks`, `blockChanged`, `copyPrev` and `frameFPS`. Their combination is consistent with repeated capture and block-level frame differencing. The Windows bundle includes `node-screenshots`, its native capture module, `jpeg-js`, MessagePack and WebSocket dependencies. Installer checks reject a bundle missing required capture files. A task named `GoogleUpdateSupporterNode` is recorded for that component. [S1]

These findings support a capture-streaming implementation assessment, not proof of an observed stream. Dependency presence, linked symbols and runtime reachability are different evidence types. An earlier dependency comparison suggested a capture-free Linux build; E18 explains why that conclusion is not sufficient to dismiss the later symbol evidence.

Negative searches and corrections

The tested key-listener packages and platform hook strings were not found. Go TLS symbols containing `keylog` refer to TLS key logging, not keystroke capture. The original categorical claim that keylogging was impossible is narrowed here to a bounded negative search. Likewise, absence of family names or shortener strings does not establish absence of related behavior or code lineage. [S1]

The content comparison in Exhibit E05-7 finds that the two equal-length payloads differ only in ZIP timestamps for two package-lock members inside their embedded Windows bundles. Their configuration is identical. This corrects the source note’s configuration-difference claim. The Windows bundle also includes synthetic input injection and clipboard modules; its capture/transport dependency list is not a complete inventory of capabilities. Synthetic input injection is distinct from keystroke capture.

Artifact identity

The examined Go implant has SHA-256 `ef183bbb40f4c7c3f4b52994cdba3295f3eab2604fdad99eefa3b67b8d966e8d`. The large JavaScript sample has SHA-256 `694626bed2df1f0d0026a3f7ead7692b44c7e73041cdd799b79a025335533cdc`. [S2]

Exhibits

`<ARTIFACT>` in a command means the artifact named in that exhibit. Byte offsets come from `strings -t d` or `grep -b`, are decimal and zero-based. Nothing was executed, and nothing was unpacked to disk: archive members were listed or streamed with `unzip -l`, `unzip -v` and `unzip -p`.

Exhibit E05-1. `.kdbx` in the private-key extension set (`KEY_EXTENSIONS`)

Artifact: Large JavaScript stage-three payload (reference sample) · 1140214 bytes · SHA-256 `694626bed2df1f0d0026a3f7ead7692b44c7e73041cdd799b79a025335533cdc`

Location: line 10 and lines 315–331

Extraction: `sed -n '10p;315,331p' <ARTIFACT>`

Supports: `.kdbx` sits in a private-key extension set that collection predicates use

```
const KEY_EXTENSIONS = new Set([".key", ".pem", ".asc", ".pfx", ".p12", ".ppk", ".gpg", ".kdbx"]);
function matchesPrivateKeyFile(name) {
  const base = path.basename(String(name || ""));
  if (SSH_KEY_FILENAMES.has(base)) return true;
  const ext = extOf(name);
  return KEY_EXTENSIONS.has(ext);
}

function matchesCredentialNamedSecretFile(name) {
  if (matchesPrivateKeyFile(name)) return true;
  if (!matchesCredentialFilename(name)) return false;
  const ext = extOf(name);
  return (
    KEY_EXTENSIONS.has(ext)
    || TEXT_EXTENSIONS.has(ext)
    || PAT_EXTENSIONS.has(ext)
    || matchesEnvFileName(name)
  );
```

`KEY_EXTENSIONS` is also used by `matchesSystemImpFile` (line 350) and `matchesDataDriveFile` (line 362). Line 10 is byte-identical in the second large payload: SHA-256 `5498038886bab8a8db2159495c015f26e322708fc7a7f3ea698e7b7c06e3d148`, line 10 (E05-6).

Exhibit E05-2. No KeePass-specific code, and where the stray `kdbx` hits are

Artifacts: the stage-three payload (`694626be`…), the second large payload (`54980388`…), the dropper `install_guard.js` · 1521039 bytes · SHA-256 `d24df1cacc01adee4e031b0f4d582448b7462f31975b57f553b7e0e86dddb184`, and the deepfake_guard stealer · 230621 bytes · SHA-256 `70ea6e2eaab58bdb83b7b235dd6253ffbcaed5e7df0286cbefd72d7650c3f47a`

Location: whole-file counts

Extraction: `` `printf "%s:" <NAME>; for p in keepass 1password onepassword bitwarden lastpass dashlane kdbx; do printf " %s=%s" $p "$(grep -a -o -i -F $p <ARTIFACT> | wc -l)"; done; printf ' ".kdbx"=%s\n' "$(grep -a -o -F '".kdbx"' <ARTIFACT> | wc -l)"` `(run once per file)

Supports: no KeePass parser or unlock in the code; other password-manager names are absent; chance matches inside encoded dropper content are excluded

```
stage3.js: keepass=0 1password=0 onepassword=0 bitwarden=0 lastpass=0 dashlane=0 kdbx=1 ".kdbx"=1
aazdias_stealer.decrypted.js: keepass=0 1password=0 onepassword=0 bitwarden=0 lastpass=0 dashla\
ne=0 kdbx=1 ".kdbx"=1
install_guard.js: keepass=0 1password=0 onepassword=0 bitwarden=0 lastpass=0 dashlane=0 kdbx=2 \
".kdbx"=0
install_guard_d.stealer.decrypted.js: keepass=0 1password=0 onepassword=0 bitwarden=0 lastpass=\
0 dashlane=0 kdbx=0 ".kdbx"=0
```

Lines are hard-wrapped at 94 characters with a trailing `\`. The two `kdbx` hits in `install_guard.js` are substrings of encoded data, not the quoted extension. Only the two large payloads contain the extension literal.

Exhibit E05-3. The three `keylog` matches in the Go implant are TLS key logging

Artifact: Go implant (linux-amd64, stripped, Go build ID present) · 15790242 bytes · SHA-256 `ef183bbb40f4c7c3f4b52994cdba3295f3eab2604fdad99eefa3b67b8d966e8d`

Location: string offsets 7718364, 7725660, 9610960

Extraction: `strings -a -t d -n 6 <ARTIFACT> | grep -i keylog`

Supports: Go TLS symbols containing `keylog` refer to TLS key logging, not keystroke capture

```
7718364 writeKeyLog
7725660 KeyLogWriter
9610960 crypto/tls.(*Config).writeKeyLog
```

`KeyLogWriter` is a field of the standard library's `crypto/tls.Config` and `writeKeyLog` is its method, which is what the third, fully qualified name shows. Blog post 04 quotes these three names, and this is their primary source.

Exhibit E05-4. Negative search for key-listener packages and platform hooks

Artifacts: Go implant (`ef183bbb`…, 15790242 bytes) and Windows agent bundle (ZIP, 192 entries) · 603347 bytes · SHA-256 `be5859f23f0781242d854dc24c102c694dabfc14a159b56429e313f0521141b6`

Location: whole-file counts (the ZIP is streamed with `unzip -p`)

Extraction: `for p in node-global-key-listener iohook uiohook robotjs SetWindowsHookEx WH_KEYBOARD GetAsyncKeyState XRecord XQueryKeymap XGrabKey /dev/input evdev CGEventTap IOHIDManager; do printf "%s=%s " $p "$(grep -a -o -F -- $p <ARTIFACT> | wc -l)"; done` (for the ZIP: `unzip -p <ARTIFACT> | grep …`)

Supports: the tested key-listener packages and platform hook strings were not found

```
Go implant:
node-global-key-listener=0 iohook=0 uiohook=0 robotjs=0 SetWindowsHookEx=0 WH_KEYBOARD=0 GetAs\
yncKeyState=0 XRecord=0 XQueryKeymap=0 XGrabKey=0 /dev/input=0 evdev=0 CGEventTap=0 IOHIDManag\
er=0
Windows agent bundle (all members):
node-global-key-listener=0 iohook=0 uiohook=0 robotjs=0 SetWindowsHookEx=0 WH_KEYBOARD=0 GetAs\
yncKeyState=0 XRecord=0 XQueryKeymap=0 XGrabKey=0 /dev/input=0 evdev=0 CGEventTap=0 IOHIDManag\
er=0
```

The `Go implant:` and `Windows agent bundle (all members):` headings were added; the count lines are verbatim. The same loop on the stage-three payload also gives 0 for every pattern. These searches are case-sensitive. A case-insensitive `xrecord` search hits once in the Go binary, inside `modernc.org/sqlite/lib._sqlite3BtreeMaxRecordSize` (offset 9784748), so it is not a hook. The Windows bundle's `lib/input-host.ps1` does call `SendInput` (3 hits). That is synthetic input injection for remote control, not keystroke capture.

Exhibit E05-5. Capture package symbols in the Go implant

Artifact: Go implant · 15790242 bytes · SHA-256 `ef183bbb40f4c7c3f4b52994cdba3295f3eab2604fdad99eefa3b67b8d966e8d`

Location: string offsets as listed

Extraction: `strings -a -t d -n 6 <ARTIFACT> | grep -E 'agent/capture(\.(CaptureDisplay|CaptureAndSend|Loop|encodeJPEG|encodeBlocks|blockChanged|copyPrev|frameFPS|avgBytes|avgMs|supportsCapture)$|/[a-z_]+\.go$)'`, plus `grep -a -o -b -E 'screenshot: no active displays found|screen capture failed on all displays|screenshot: skipped \(screen capture disabled\)' <ARTIFACT>`

Supports: `CaptureAndSend`, `Loop`, `encodeBlocks`, `blockChanged`, `copyPrev`, `frameFPS`; repeated capture with block-level frame differencing

```
10271729 overlord-client/cmd/agent/capture.Loop
10271768 overlord-client/cmd/agent/capture.supportsCapture
10272273 overlord-client/cmd/agent/capture.CaptureAndSend
10272322 overlord-client/cmd/agent/capture.avgMs
10272362 overlord-client/cmd/agent/capture.avgBytes
10272868 overlord-client/cmd/agent/capture.frameFPS
10273194 overlord-client/cmd/agent/capture.copyPrev
10273237 overlord-client/cmd/agent/capture.encodeBlocks
10273333 overlord-client/cmd/agent/capture.blockChanged
10273478 overlord-client/cmd/agent/capture.encodeJPEG
10285920 overlord-client/cmd/agent/capture.CaptureDisplay
10408969 overlord-client/cmd/agent/capture/capture_linux.go
10409020 overlord-client/cmd/agent/capture/capture.go
10409065 overlord-client/cmd/agent/capture/jpeg_std.go
8970207:screenshot: no active displays found
8973833:screen capture failed on all displays
8999219:screenshot: skipped (screen capture disabled)
```

This is the primary source for `capture.CaptureDisplay` and `capture.encodeJPEG` as quoted in blog post 04. The Linux-specific source file `capture_linux.go` is linked in (see E18). These are symbol and string names only. They do not prove that the capture code runs.

Exhibit E05-6. Windows capture bundle: declared dependencies and the mandatory-bundle check

Artifacts: Windows agent bundle (ZIP) · 603347 bytes · SHA-256 `be5859f23f0781242d854dc24c102c694dabfc14a159b56429e313f0521141b6`; stage-three payload · 1140214 bytes · SHA-256 `694626bed2df1f0d0026a3f7ead7692b44c7e73041cdd799b79a025335533cdc`

Location: ZIP member `package.json`, lines 5–14; stage-three payload, lines 7032 and 7044–7051

Extraction: `unzip -p <ZIP> package.json | sed -n '5,14p'`; `sed -n '7032p;7044,7051p' <STAGE3>`

Supports: `node-screenshots`, `jpeg-js`, MessagePack and WebSocket dependencies; installer rejects a bundle without the capture files; task `GoogleUpdateSupporterNode`

```
   "description": "Hidden Windows Node WebSocket agent (Overlord server parity)",
   "main": "update-agent.js",
   "scripts": {
     "prepare-bundle": "npm install --omit=dev",
     "start": "node update-agent.js"
   },
   "dependencies": {
     "@msgpack/msgpack": "^3.1.2",
     "jpeg-js": "^0.4.4",
     "node-screenshots": "^0.2.8",
--- stage-three payload ---
const WIN_NODE_AGENT_TASK = "GoogleUpdateSupporterNode";
function winNodeAgentVerifyCaptureBundle(root) {
  if (!fs.existsSync(winNodeAgentCaptureNativePath(root))) {
    throw new Error("capture native module missing from bundle");
  }
  if (!fs.existsSync(path.join(root, "node_modules", "node-screenshots", "index.js"))) {
    throw new Error("node-screenshots package missing from bundle");
  }
}
```

The `--- stage-three payload ---` separator was added. The next `package.json` line (15) is `"ws": "^8.18.3"`. `unzip -l` lists the native module `node_modules/node-screenshots-win32-x64-msvc/node-screenshots.win32-x64-msvc.node` at 700,928 bytes. The base64 constant `WIN_NODE_AGENT_BUNDLE_B64` (line 7031) in the stage-three payload decodes to exactly this ZIP (SHA-256 `be5859f2…`, see E05-7).

Exhibit E05-7. Content comparison of the two equal-length payloads

Artifacts: stage-three payload · 1140214 bytes · SHA-256 `694626bed2df1f0d0026a3f7ead7692b44c7e73041cdd799b79a025335533cdc`; second large payload · 1140214 bytes · SHA-256 `5498038886bab8a8db2159495c015f26e322708fc7a7f3ea698e7b7c06e3d148`; Windows agent bundle (`be5859f2…`)

Location: line 7031 of both payloads (bytes 301941–1106441); ZIP local header at ZIP offset 502270

Extraction: `cmp -l <A> <B> | wc -l`; `cmp <A> <B>`; `` `sed -n '7031p' <X> | sed -E 's/^const WIN_NODE_AGENT_BUNDLE_B64 = "//; s/";?$//' | base64 -d | sha256sum (decoded to a pipe only); xxd -s 502270 -l 48 <ZIP>` ``

Supports: "Equal byte length and different hashes … do not prove that configuration is their only difference" (here the comparison is made)

```
43
<A> <B> differ: byte 971682, line 7031
be5859f23f0781242d854dc24c102c694dabfc14a159b56429e313f0521141b6  -     (decoded from <A>)
8a36b018a002c3cf19a1945f091ea2487cdf9e6ceb3d08b6571d0adf67339534  -     (decoded from <B>)
0007a9fe: 504b 0304 1400 0000 0800 f0a9 f45c bda5  PK..........\..
0007aa0e: fae0 b203 0000 2f0b 0000 1f00 1c00 6e6f  ....../.......no
0007aa1e: 6465 5f6d 6f64 756c 6573 2f2e 7061 636b  de_modules/.pack
```

The `(decoded from …)` labels were added, and `cmp` printed real file names where the block shows `<A>` and `<B>`. All 43 differing bytes fall in four runs (starting at bytes 971682, 1032426, 1100836 and 1103870), all inside the embedded base64 ZIP. In the decoded ZIPs, 26 bytes differ. They are the DOS time/date fields and the `UT` timestamp extra fields of the local and central headers for `node_modules/.package-lock.json` and `package-lock.json`. The CRC-32 values are identical (`bda5fae0` above). Everything else is byte-identical, including the embedded agent token and the server settings.

Claims without a retained primary artifact

- The statement that 18 decrypted or text samples were examined covers files spread over five incident folders. This document exhibits only the four samples that carry findings. The negative family-name and `short.gy` searches over the other samples are not shown.
- The persistence-task properties (`Hidden`, `LogonTrigger`, `RestartOnFailure`) quoted in the re-examination note were not excerpted here. Only the task name (E05-6) was.

Exhibit inputs

Retained inputs identified by size and SHA-256. These include artifacts and analytical records; each exhibit identifies its evidence type and extraction limitations. The files themselves are not distributed.

stage-three payload · 1,140,214 bytes

`SHA-256 694626bed2df1f0d0026a3f7ead7692b44c7e73041cdd799b79a025335533cdc`

Installer JavaScript · 1,521,039 bytes

`SHA-256 d24df1cacc01adee4e031b0f4d582448b7462f31975b57f553b7e0e86dddb184`

Decrypted deepfake-guard collector · 230,621 bytes

`SHA-256 70ea6e2eaab58bdb83b7b235dd6253ffbcaed5e7df0286cbefd72d7650c3f47a`

Go implant · 15,790,242 bytes

`SHA-256 ef183bbb40f4c7c3f4b52994cdba3295f3eab2604fdad99eefa3b67b8d966e8d`

Windows agent bundle (ZIP) · 603,347 bytes

`SHA-256 be5859f23f0781242d854dc24c102c694dabfc14a159b56429e313f0521141b6`

second large payload · 1,140,214 bytes

`SHA-256 5498038886bab8a8db2159495c015f26e322708fc7a7f3ea698e7b7c06e3d148`

Source records and fingerprints

SHA-256 identifies the reviewed source bytes, not acquisition time or every artifact described. Restricted originals are not reproduced.

[S1] Static capability re-examination, 24 July 2026 · 9,290 bytes

`SHA-256 75b0ac85d0d6dd6d7b68588a0618252369a5431c268e8f4b04f24a2f4d9f3117`

[S2] Companion validation record, 21 October 2026 · 2,562 bytes

`SHA-256 613d16e0bed012c96e00a7db6e2f7260b263fa5cd3941d39d523a71dfc94a74d`