E04. Authenticated payload-decryption analysis

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.

Material and recorded procedure

The analysis record dated 22 July 2026 describes recovery of three encrypted payloads from the VSIX through static analysis. An Acorn parse identified decoder wrappers and the expression constructing an AES-256 key from sixteen fragments: nine decoder-wrapper calls and seven literals. String-array rotation was resolved across multiple call sites before decoding. The target JavaScript was not imported or executed as part of that procedure. [S1]

The recorded encrypted layout is a 12-byte initialization vector, a 16-byte authentication tag and the remaining ciphertext. The record reports successful AES-GCM authentication for all three payloads using a cryptographic library. This is stronger than judging the plaintext by appearance. The decryption was not rerun in preparing this document, so the authentication result remains a reported case result rather than a new independent verification. Key material is omitted.

Recovered payload identities

- `detect_malware.py`: 60,054 bytes; SHA-256 `4c3fdd86a59d52a0e29f472666cbf9472b6fd83c38ef2a4b547f31c72b8ee87e`.
- `windows-agent-node.js`: 56,093 bytes; SHA-256 `e5b7d6411d0c95da047185d94fca98d454b73df2e7f46f2753e212af2ca45da5`.
- `windows-js-pipeline.js`: 32,324 bytes; SHA-256 `bf820623e8deaa37092650c7fae594e56b8d81c27b541419006cca2d0536353b`.

The outputs are readable, commented Python and JavaScript. Encryption at rest and source obfuscation are separate properties. [S1]

Functional observations

The Python source describes Chromium key recovery through DPAPI and browser elevation services, with caller-path conditions for App-Bound Encryption. The pipeline coordinates local collection, elevated decryption, upload and cleanup. The Node agent implements a Windows collection route that does not require the native Go agent. These observations support platform-specific execution branches and explain why native-agent hashes alone do not cover all variants. [S1]

Limits

Comments describe intended behavior and may be incomplete or inaccurate. Authentication ties plaintext to encrypted input but does not show successful browser decryption, elevation or exfiltration. Those outcomes depend on browser version, policy, user context and host state. E19 separately assesses the elevation attempt. Distinct recovered keys establish key differences between the examined branches, not independent authorship or a universal deployment policy.

Exhibits

Exhibit E04-1. The AES-256 key-assembly expression (key value redacted)

Artifact: extension/extension.js · 606,460 bytes · SHA-256
`d5e9288693aa745dc89368deac677e7ea1ec81e663283af30838cdae189b7a7e`
Location: byte offset ~1990, the `ENC_KEY_HEX=` initializer
Extraction: byte-offset read (`dd … skip=1990`) of the key initializer; file parsed statically, never executed
Supports: "an AES-256 key from sixteen fragments" (nine decoder calls, seven plain literals)

```
ENC_KEY_HEX=_0x3ef45d(0xb59,0xea4,'9Rlw',0x771)+'[REDACTED: key fragment]'
  +_0x3ef45d(0x64d,0xba0,'k^@W',0xd2a)+'[REDACTED]'
  +(_0x2bac03(0x530,'rkdk',0xc22,0x684)+_0x3ef45d(0x35a,0x113,'%jkR',0x49f)
    +_0x2bac03(0x6f3,'%jkR',0xde9,0xe75)+'[REDACTED]')
  +('[REDACTED: key fragment]'+_0x3ef45d(0x573,0x631,'qzCC',0x442)
    +_0x2bac03(0xa45,'qxVi',0x9ef,0x73b)+'[REDACTED]')
  +(_0x3ef45d(0xa65,0x962,'bTub',0xdad)+'[REDACTED: key fragment]'
    +_0x2bac03(0x8b9,'tLeT',0xfb3,0x565)+'[REDACTED]'),
ENC_KEY=Buffer[_0x2bac03(0xac9,'y5n!',0xf50,0xcc2)](ENC_KEY_HEX,[REDACTED: 'hex'])
```

The structure is verbatim: nine decoder-wrapper calls (`_0x3ef45d` ×5, `_0x2bac03` ×4) interleaved with seven plain string literals, concatenated in source order — sixteen fragments in all. Every plain literal is a fragment of the 256-bit key, so each is marked `[REDACTED: key fragment]`; the wrapper-call arguments (array indices and RC4 key tokens) are left intact because they are not themselves key plaintext. Reformatted with line breaks for width; tokens unchanged.

Exhibit E04-2. Encrypted-blob layout constants (IV / tag / ciphertext)

Artifact: extension/extension.js · 606,460 bytes · SHA-256
`d5e9288693aa745dc89368deac677e7ea1ec81e663283af30838cdae189b7a7e`
Location: the `decrypt()` body, the three `subarray` slices applied to the input buffer
Extraction: static byte-offset read of the decrypt routine (parsed, not executed)
Supports: "a 12-byte initialization vector, a 16-byte authentication tag and the remaining ciphertext"

```
iv         = buf.subarray(0, 12)     // bytes 0..11
tag        = buf.subarray(12, 28)    // bytes 12..27  (16-byte GCM tag)
ciphertext = buf.subarray(28)        // bytes 28..end
cipher     = crypto.createDecipheriv('aes-256-gcm', KEY, iv)
             …[elided: AES-GCM setAuthTag / update / final decryption sequence]…
```

The slice boundaries (`0:12`, `12:28`, `28:`) and the `aes-256-gcm` algorithm string are the load-bearing constants, extracted verbatim from the obfuscated routine and shown here in a plain-language normalisation of the minified source. The actual `setAuthTag`/`update`/`final` decryption call sequence is elided so the excerpt is not a runnable decryptor.

Exhibit E04-3. detect_malware.py — recovered plaintext header

Artifact: detect_malware.py (decrypted VSIX payload) · 60,054 bytes · SHA-256
`4c3fdd86a59d52a0e29f472666cbf9472b6fd83c38ef2a4b547f31c72b8ee87e`
Location: lines 1-14 (module docstring)
Extraction: `sed -n '1,14p' detect_malware.py`
Supports: "readable, commented Python"; Chromium key recovery via DPAPI and App-Bound Encryption

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Decrypt Chrome / Edge saved passwords from a Login Data SQLite DB on the SAME Windows user
profile, using the AES master key from Local State.

Requires: pip install pycryptodome

Key material:
  - Older Chromium: os_crypt.encrypted_key -> DPAPI -> 32-byte AES key.
  - Current Chrome / Edge (App-Bound Encryption): os_crypt.app_bound_encrypted_key must be
    decrypted via the browser Elevation Service (COM). DPAPI-only key no longer matches
    password blobs, so decryption fails with AES-GCM MAC errors if ABE is ignored.
```

First lines of the decrypted payload, verbatim. The plaintext confirms the file decrypted and is the browser-credential decryptor the report describes. No redaction needed in this span.

Exhibit E04-4. windows-js-pipeline.js — recovered plaintext header

Artifact: windows-js-pipeline.js (decrypted VSIX payload) · 32,324 bytes · SHA-256
`bf820623e8deaa37092650c7fae594e56b8d81c27b541419006cca2d0536353b`
Location: lines 1-10 (module header comment)
Extraction: `sed -n '1,10p' windows-js-pipeline.js`
Supports: "pipeline coordinates local collection, elevated decryption, upload and cleanup"; no Go dependency

```
/**
 * Windows default pipeline (extension host): runs Node.js agent scripts for
 * phase1 + elevated phase2 (credential decryption via detect_malware.py),
 * POSTs artifacts to company-wallet API, then cleans up vendor/.vscode/runtime copies.
 *
 * No Go exe dependency -- all logic is in windows-agent-node.js + detect_malware.py.
 *
 * Config: googleUpdateSupport.* workspace settings or package.json googleUpdateSupport
 *         (patched from repo .env via patch-project).
 */
```

Verbatim header of the decrypted coordinator. Establishes the phase1/phase2 model and the Go-free Windows branch. No redaction.

Exhibit E04-5. windows-agent-node.js — recovered plaintext header

Artifact: windows-agent-node.js (decrypted VSIX payload) · 56,093 bytes · SHA-256
`e5b7d6411d0c95da047185d94fca98d454b73df2e7f46f2753e212af2ca45da5`
Location: lines 1-14 (module header comment)
Extraction: `sed -n '1,14p' windows-agent-node.js`
Supports: "Node agent implements a Windows collection route that does not require the native Go agent"

```
/**
 * Node.js replacement for the Windows Go exe agent.
 *
 * Phase 1 (--company-wallet-phase1-local): Zips browser data files (Login Data,
 * Local Extension Settings, IndexedDB, Firefox extension storage, CLI wallet)
 * into a _phase1.zip in the artifacts directory. No elevation required.
 *
 * Phase 2 (--company-wallet-job): Builds a PowerShell script that calls
 * detect_malware.py for credential decryption (DPAPI + app-bound COM + SQLite),
 * then creates a _pass.zip with e_p.txt + agent log. Runs elevated.
 *
 * No npm dependencies -- uses only Node.js built-in modules + Windows system tools.
 */
```

Verbatim header, confirming the all-JavaScript Windows collection stage. No redaction.

Claims without a retained primary artifact

- "Successful AES-GCM authentication for all three payloads": per the brief's static-only rule, decryption was not re-run here, so the authentication-pass result remains a reported case result. It is strongly corroborated by the retained plaintext payloads (E04-3/4/5), whose SHA-256 values match the report exactly, and by the key-assembly and blob-layout code (E04-1/2).
- "The two delivery vectors use different keys": the ClickFix-chain key is documented in another incident's report, not reproduced here; only the VSIX-vector key expression is a local artifact.

Encrypted-copy identity

The retained encrypted copies differ: the `.enc` blobs inside the VSIX (the extracted extension's `*.enc` files) and the separately held `*.enc` copies have different SHA-256 values despite decrypting to the same plaintext — consistent with AES-GCM using a fresh IV per encryption. The report says the payloads are "present both inside the VSIX and beside it," which does not assert byte-identity.

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.

extension/extension.js (obfuscated loader, key-assembly + blob layout) · 606,460 bytes

`SHA-256 d5e9288693aa745dc89368deac677e7ea1ec81e663283af30838cdae189b7a7e`

detect_malware.py (decrypted VSIX payload, Chrome ABE decryptor) · 60,054 bytes

`SHA-256 4c3fdd86a59d52a0e29f472666cbf9472b6fd83c38ef2a4b547f31c72b8ee87e`

windows-js-pipeline.js (decrypted VSIX payload, coordinator) · 32,324 bytes

SHA-256 bf820623e8deaa37092650c7fae594e56b8d81c27b541419006cca2d0536353b

windows-agent-node.js (decrypted VSIX payload, Windows collection agent) · 56,093 bytes

SHA-256 e5b7d6411d0c95da047185d94fca98d454b73df2e7f46f2753e212af2ca45da5

Source records and fingerprints

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

[S1] Authenticated payload-decryption record, 22 July 2026 · 9,192 bytes

SHA-256 b34d0468651a78af7816d70d308e385b5e722824fec268a833532fe8a1375d7f