From Delivery To Execution: An Evasive Azorult Campaign Smuggled Through Google Sites

By Jan Michael Alcantara

Published: 2024-03-15 · Archived: 2026-04-05 12:50:10 UTC

Summary

Netskope Threat Labs has observed an evasive Azorult campaign in the wild that employs multiple defense evasion techniques from delivery through execution to fly under the defender's radar as it steals sensitive data.

Azorult is an information stealer first discovered in 2016 that steals sensitive information including user credentials, browser information, and crypto wallet data. Azorult is on the rise and is currently one of the top malware families that Netskope Threat Labs has observed targeting the healthcare industry over the last year.

In this blog post, Netskope Threat Labs performs a detailed teardown of an evasive AzoruIt malware campaign we observed in the wild. This campaign is noteworthy for the following reasons:

- It delivers its initial payload through HTML smuggling, a detection evasion technique that is gaining popularity among adversaries. This defense evasion technique was also used by a nation-state group to smuggle a remote access trojan, and by Nokoyawa ransomware, where they started the infection process through HTML smuggling.
- It uses an unorthodox HTML smuggling technique where the malicious payload is embedded in a separate JSON file hosted on an external website.
- It executes the fileless Azorult infostealer stealthily by using reflective code loading, bypassing disk-based detection and minimizing artifacts.
- It uses an AMSI bypass technique to evade being detected by a variety of host-based anti-malware products, including Windows Defender.
- It steals sensitive data, including information for 137 distinct crypto wallets, login credentials, browser files, and important documents.

Google Sites serves as a decoy for HTML smuggling

HTML smuggling is a defense evasion technique that aims to bypass web controls that block risky file types. It abuses legitimate HTML5 download attributes and Javascript blobs to construct malicious payloads on the client side, bypassing network security filters.

As part of Netskope Threat Labs' threat hunting activities, we uncovered a campaign wherein an attacker created fake Google Docs pages on Google Sites from which they used HTML smuggling to download malicious payloads. They lure their victims to the fake Google Docs pages to trick them into believing the downloaded file was from Google Docs. In most cases that we see in the wild, the adversary embeds the smuggled malicious payload in the Javascript itself. In this example, the adversary embedded the malicious payload in a separate

JSON file as a BASE64 encoded string. When the victim accesses the website, it sends a GET request to download the JSON file from a separate domain (`mahmudiyeresort[.]com[.]tr`) and extracts the payload from there.

HTML smuggling code that collects Payload From Compromised Domain

Smuggling With A Captcha

Usually, when a victim accesses a website that uses HTML smuggling to deliver malicious payloads, the payload is downloaded immediately. For this campaign, the attacker's website hosted on Google Sites initiates a CAPTCHA, which serves as an additional layer of protection against URL scanners. This helps the HTML file to remain undetected in public scanners like Virustotal, which cannot proceed past the CAPTCHA.

HTML smuggling code that collects Payload From Compromised Domain

Malicious shortcut downloading multiple Powershell and Javascripts

Once the user passes the CAPTCHA test, the HTML smuggling Javascript code reconstructs the payload and downloads it to the victim's machine. The payload is an LNK shortcut file that uses a PDF icon to trick users into clicking it. Clicking on the LNK file kicks off the following chain of events.

1. The LNK file spawns a command prompt where it saves a base64 encoded Powershell command to a batch file named `Fyap4cKJ.bat`
2. The Powershell command is then decoded using a Windows native application named `certutil.exe` and overwrites `Fyap4cKJ.bat`
3. It then creates a scheduled task named `t09pxsrXKG` that executes the batch file `Fyap4cKJ.bat`.
4. The batch file `Fyap4cKJ.bat` will execute a Powershell script that uses `Invoke-WebRequest` to download a PHP file from `sqjeans[.]com` and saves it as `qtoW0vI2.js` in the temp folder. It will then execute `qtoW0vI2.js` using `wscript.exe`.

5. The Powershell command from the batch file `Fyap4cKJ.bat` then deletes the scheduled task `t09pxsrXKG` created earlier.

LNK payload with PDF icon

The Javascript `qtoW0vI2.js` then performs the following three tasks:

1. Copies itself in the `%ProgramData%`, and renames itself as `agent.js`
2. Checks and deletes itself if a file named `7z52OJFPXT4J` exists in the temp folder.
3. Downloads two Powershell scripts named `agent1.ps1` and `agent3.ps1` using `Invoke-WebRequest`, and executes them using `Invoke-Expression`.

qtoW0vI2.js from the compromised domain

Azorult Fileless Malware Loaded Through Reflective Code Loading

Another defense evasion technique the attacker uses is to execute the Azorult infostealer in memory using reflective code loading. Reflective code loading of a portable executable file means that instead of writing and running the malware on disk where it leaves more footprints and artifacts, it loads code into a running Powershell process's own memory. Let's look at how they accomplish reflective code loading with the two Powershell scripts executed.

agent1.ps1

The first powershell script (`agent1.ps1`) executed is used to bypass the Antimalware Scan Interface (AMSI). It does so by setting `AmsiInitFailed` to a `True` value so that AMSI initialization fails, which means no scan will be performed for the current process.

The value of `AmsiInitFailed` is defined by the Javascript `qtoW0vI2.js` as it executes `agent1.ps1` .

AMSI bypass from agent1.ps1

agent3.ps1

The second powershell script (`agent3.ps1`) is set to perform the following tasks:

1. Download an Azorult loader

The Powershell script (`agent3.ps1`) starts by downloading the Azorult loader (`service.exe`) from the earlier compromised domain using `Invoke-WebRequest` . The executable was likely compiled in November 2023 and was first submitted to VirusTotal in February.

Azorult Loader General Information using Detect It Easy

The loader downloaded is not actually written on disk but is later executed in an allocated memory. The binary content is stored as a byte array in the variable `$image`, and later stored in a memory block buffer `$imagebBuf` using a copy function from `System.Runtime.InteropServices.Marshal`.

Powershell script storing the loader into byte array and later allocated into a memory block

The loader contains some anti-analysis features where it terminates its process when it matches its list of common sandbox usernames and hostnames. The following usernames and hostnames are monitored: “Paul Jones”, “Joe Cage”, “PJones”, “STRAZNJICA GRUBUTT”, “WillCarter-PC”, “FORTI-PC”.

Loader terminating process based on common sandbox username and hostname

2. Define a shellcode

A shellcode on the byte array `$sc` is defined on the script. Later on, this will be executed in memory within the same thread of the downloaded executable.

Shellcode loaded along with the initially loaded malware

3. Execute a routine that loads both shellcode and executable into process memory

The script defines two functions: `Get-DelegateType` (GDT) and `Get-ProcAddr` (GPA). The GDT function defines a delegate type at runtime, while the GPA function returns a function's memory address from a module. Later in the script, the GPA function gets the memory addresses of `VirtualAlloc`, `CreateThread`, and `WaitForSingleObject` from `kernel32.dll`.

The script then initiates memory allocation for the shellcode (`$sc`) by invoking `VirtualAlloc`. Subsequently, the content of the shellcode is transferred to the allocated memory represented by variable `$x` using the `Marshal.Copy` method.

As for the Azorult loader (`service.exe`), the script employs Marshal's `AllocHGlobal` method to allocate unmanaged memory, sized according to the length of the executable's byte array (`$image`), and assigns it to `$imageBuf`. Subsequently, the binary content of the executable stored in `$image` is transferred to the allocated memory buffer `$imageBuf` using the `Marshal.Copy` method

Lastly, the script uses `CreateThread` to initiate a new thread, providing the memory addresses of both the shellcode and the executable. It then employs `WaitForSingleObject` to await the completion of the thread's execution process.

Loading of of shellcode and executable in memory

sd2.ps1

When the loader is executed, it sends an HTTP GET request to download and execute another Powershell script named `sd2.ps1`. The script initially connects to a C2 server to collect an XOR key and store it in the `$config` variable. The key is then used for a byte-wise XOR operation for the Azorult binary defined inside the script.

sd2.ps1

Azorult

The Azorult payload is a .NET compiled binary named `pg20.exe` executed in memory using the `Assembly.Load` method. It aims to collect sensitive user information, including personal documents, crypto wallet information, login credentials, and browser data. All stolen files and data are then transmitted to the C2 server over HTTP.

The malware starts with a parameter check. If fewer than 2 parameters are passed, it terminates execution.

Parameter check with a dummy C2 server

Afterward, Azorult proceeds to generate a private and public key and a shared secret. It uses Curve25519 elliptic curve cryptography to generate a 32-byte private key, and from it generate a public key and shared secret. These are used to encrypt the stolen files to prevent detection as the stolen files get uploaded over to the C2 server.

Routine to generate private and public keys

The infostealer proceeds to collect the following data from the victim:

Primary Display Screenshot

Azorult initially collects a screenshot of the device's screen. It uses the `GetDeviceCaps` method to retrieve the resolution of the primary display screen. It then captures a screenshot of the entire screen using the `Graphics.CopyFromScreen` method, which copies the screen's content onto a bitmap object. Finally, the captured image is saved as a JPEG file.

Code to collect screenshot

Browser Data

Azorult proceeds to pilfer the victim's browser data. It copies the content of Chrome's Login Data, Local State, Cookies, and Web data into the roaming folder path to be exfiltrated later. It uses the `InternalCopy` method internally defined, using the Win32 `CopyFile` function to copy all the browser data.

Routine to copy Chrome's related files

The same routine is applied to Firefox's database files. It copies the content of several database related files and saves it to the roaming profile path. It copies the following files: `formhistory.sqlite`, `places.sqlite`, `cookies.sqlite`, `logins.json`, and `key4.db`.

Crypto wallet information

The infostealer then proceeds to copy crypto wallet data from the target machine stored in Chrome, Edge, and Firefox. The malware contains a list of 119 target Chrome wallets and 12 Edge wallet extensions. It verifies if a user has installed a Google Chrome or Edge wallet extension by checking if the wallet's folder name from the list is present in the Local Extension directory. If the wallet directory is present, it will copy its files to the roaming profile directory.

Routine to verify and copy Chrome's wallet extension

Target Chrome Wallets

Target Edge Wallets

Similarly, Azorult also looks for some target wallet extensions on Firefox. It does so by collecting all Firefox extension IDs on the device and cross checking it with the target wallet list. Once the presence of the wallet extension is confirmed, it copies the contents of the file to the roaming profile path.

Routine that looks for Crypto wallet Firefox extension

Target Firefox Wallets

Sensitive Documents

Lastly, Azorult looks for sensitive documents and files on the Desktop. It searches for certain file extensions and files named with keywords that might contain personal sensitive information. While searching for specific files, it also skips certain file types.

Target File Extension

txt	axx	doc	xls
kdbx	docx	xlsx	

File Name Keywords

backup	wallet	ledger	safepal	UTC–20*

two-fa	secret	trezor	paper-wallet-*.png	

Unwanted File extension

lnk	js	cpp	d
exe	ts	h	pdb
dll	asm	php	svg
obj	s	py	wav
vcxproj	c	pyc	smali
vcproj	cc	cs	

When all conditions are met, it reads the contents of the files and writes them to a memory stream, which is later used to exfiltrate to the C2 server.

Routine to copy the content of file

All files and data collected are sent to the server via HTTPS using the `WebRequest` class. The data copied are compressed using GZip compression and then stored in an array. The array is then encrypted using the earlier created shared secret, then sent over to the C2 server using the `WebRequest.Create` method. The public key generated earlier will be sent over through `requestStream`.

Routine to exfiltrate stolen file with a dummy C2 server

Sample exfiltrated file

Conclusions

Azorult is an infostealer used to steal user's credentials and card information. In this blog, we analyzed a campaign wherein an attacker used multiple evasion techniques, including HTML smuggling, bypassing of AMSI, and reflective code loading, to hide their activity from security controls. Unlike common smuggling files where the blob is already inside the HTML code, this campaign copies an encoded payload from a separate compromised site. Using legitimate domains like Google Sites can help trick the victim into believing the link is legitimate. The Azorult malware then pilfers multiple sensitive information and payment data from its target. Netskope Threat Labs will continue monitoring this and other malware campaigns used to spread infostealers.

IOCs

All the IOCs related to this campaign can be found in our GitHub repository.
