DEVCORE

The Art of Cyber Craftsman
紅隊該如何打造自己的利器

Inndy

戴夫寇爾股份有限公司
inndy@devco.re

DEVCORE CONFERENCE 2024 | 2024.03.16

講者介紹

DEVCORE

- Inndy
- 專職紅隊工具開發
- 曾經在藍隊做研究員，天天與 APT 以及惡意程式打交道
  - 與團隊一起投稿上過 Black Hat USA，揭露針對半導體產業的 APT 攻擊
  - 有時我會思考這些惡意程式哪裡可以做得更好...

為什麼有今天這場演講？

DEVCORE

Outline

DEVCORE

- 滲透工具的選擇與開發策略
- 對抗防禦產品與偵測
- 攻擊者的風險
- 學習與精進方向

身為滲透測試工程師你可能遇過...

- 打下官網 IIS 上傳 Web Shell
- 上傳提權工具
- Mimikatz dump hash
- 沒抓到有用的 Credential
- 開始執行 SSH Bruteforce
- 防毒軟體刪掉 Web Shell
- 又被防毒軟體刪掉了 QQ
- 這次你用了珍藏免殺加殼版
- 哇，做白工總是人生必經之路
- 結果被內網 IPS 發現

DEVCORE

身為滲透測試工程師你可能遇過...

- 從設定檔找到一組可用帳號密碼
- Psexec 橫向移動到內網機器上
- UAC Bypass
- 跑 SharpHound
- ……

- 終於有點成果？
- 500KB 300KB 投放成功！
- 你嘗試了三種 Exploit 才成功
- EDR 發出告警
- BAD END

DEVCORE

剛剛的過程你被偵測了幾次？

DEVCORE

有多少工具上傳後就被防毒刪掉了？

DEVCORE

你花了多少時間幫工具改字串、加殼？

DEVCORE

DEVCORE

紅隊工具的選擇

工具的選擇策略

DEVCORE

	商業工具	開源工具	自製工具
優勢	售後服務與文件 驗證與測試	選擇眾多 各種新技術	可控性高 量身訂做
劣勢	出口管制 相對容易被偵測	品質不一 容易被偵測	占用人力資源 技術門檻
成本	$$$	踩地雷	開發時間

DEVCORE

開發方向與方法

如何選擇語言

DEVCORE

- 開發速度、難易度
- 程式執行速度
- 語言特性與程式性質
  - 同一份程式碼的跨平台能力
- 可掌控底層細節的程度
- 現有參考資料、開源程式碼
- 後加工難易度 (變形、免殺、加殼)
  - 對抗分析的潛力

~~以下觀點受到講者偏好影響，請斟酌參考~~

DEVCORE

C/C++

DEVCORE

- 優勢
  - 原生系統 API 都是 C 語言介面，相容性最好
  - 各種作業系統都能找到 C 編譯器
  - 編譯後執行速度快
- 劣勢
  - 需要掌握太多細節，並且容易發生記憶體相關的錯誤
  - 開發速度慢，並且標準程式庫薄弱
  - C++ 語言規格複雜，掌握難度高

.NET (C#, VB.NET)

DEVCORE

- 優勢
  - 標準程式庫與第三方套件豐富
  - 開發與執行速度相對快
- 劣勢
  - Windows 為主要戰場，.NET Core 編譯後可跨平台使用
  - 容易逆向與分析，但有眾多混淆方案
  - 不易掌握細節底層的行為，要使用 unsafe 與 Marshal

PowerShell

DEVCORE

- 優勢
  - 可輕易使用 .NET 內建程式庫與其他套件
  - 不需要編譯器，從 Windows 7 開始內建
  - 與 Windows 整合度高，可避免呼叫外部命令列工具
- 劣勢
  - 語法風格奇特，語言特性難以掌握
  - 只能用在 Windows（沒有人想在 Linux 與 macOS 上安裝 PowerShell）
  - 要對抗 AMSI (Anti Malware Scan Interface)

其他腳本語言 (Python, Ruby, JS, Perl)

DEVCORE

- 優勢
  - 第三方套件豐富
  - 開發速度快

其他腳本語言 (Python, Ruby, JS, Perl)

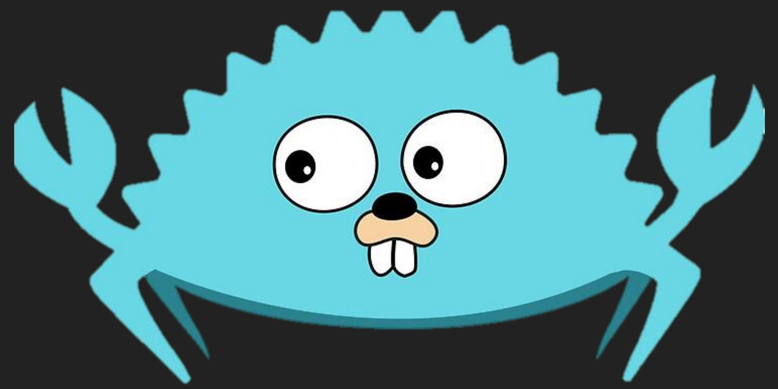
DEVCORE

- 劣勢
  - 執行速度慢
  - 缺乏底層細節掌握能力，需透過 FFI (如 ctypes)
  - 跨平台能力有限，標準程式庫缺乏的能力需依賴第三方套件
  - 目標環境不一定有對應直譯器，或者版本老舊
  - 執行受限於直譯器，缺乏有效混淆手段

新興程式語言 (Golang, Rust)

DEVCORE

- 優勢
  - 開發速度快、執行速度快
  - 第三方套件豐富
  - 底層細節掌握程度高
- 劣勢
  - Golang 編譯後檔案體積龐大
  - Rust 學習成本高
  - 語言使用者相對少



DEVCORE

對抗防禦產品

防禦機制對抗

DEVCORE

- 偵測機制是怎麼設計的？
  - 靜態特徵、動態行為
- 現代 EDR 或防毒軟體怎麼處理未知程式？
  - 第一次看見時封鎖
  - 依照程式連線目標、Child Process 自動判斷
- 被偵測的原因是什麼？可以怎麼對抗或繞過？

防禦產品常見工作流程

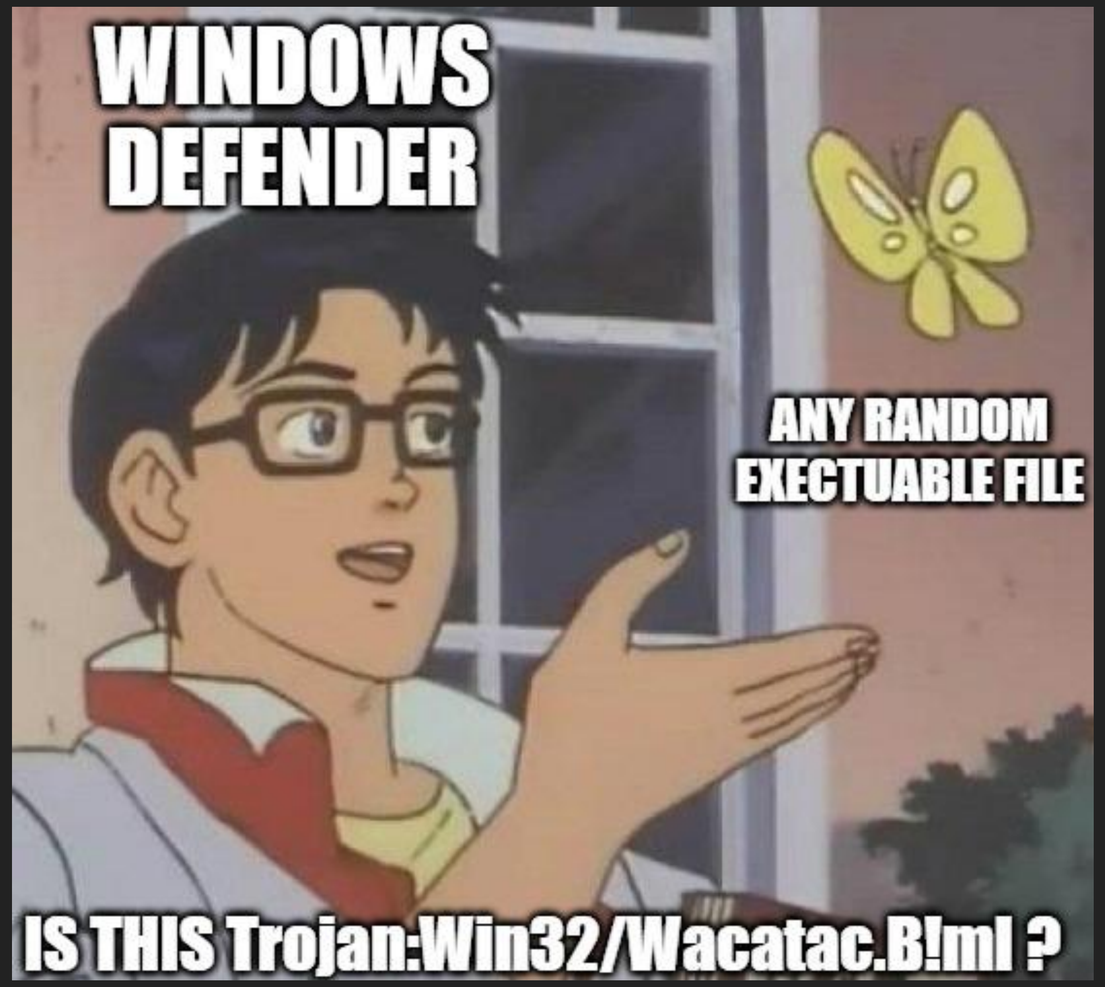
DEVCORE

- 程式執行以前
  - 檔案新增、修改觸發靜態特徵掃描
  - 執行檔第一次執行時用 Sandbox 在背景模擬一次
  - 紀錄執行檔路徑、執行參數、檔案 Hash
- 程式執行以後
  - 注入 DLL 進行 API Hooking，觀察動態行為
  - 執行前期對記憶體內容進行詳細掃描，後續定期掃描記憶體的比對特徵較少

為什麼會被防毒軟體偵測？

DEVCORE

- 使用到特定 API 與參數的組合就會被偵測
  - WriteProcessMemory
  - CreateRemoteThread
- 某些編譯器編譯 Hello World 都會被偵測
  - Golang, MinGW 都發生過
- 字串內容被當作特徵
  - vssadmin delete shadows
- Trojan:Win32/Wacatac.B!ml



躲避靜態特徵檢查

DEVCORE

- 混淆、加殼
  - Script / .NET 套用混淆工具，原生執行檔加殼
- 修改 .NET 專案 GUID (例如 SharpHound)
  - 尤其是開源專案
- 移除不必要的字串
  - Debug log 越詳細，逆向工程越輕鬆
  - 甚至上 GitHub 就找到原始碼了！

躲避靜態特徵檢查

DEVCORE

- 關鍵字串進行分割、編碼、加密
  - 尤其是主要行為或獨特字串
- 加入無作用程式碼
  - 讓 Import Table 看起來正常、干擾定位靜態特徵
- 開發 Loader 工具
  - 拆分成多個檔案、可拋棄式 Loader（被偵測也不痛）

對抗行為偵測

DEVCORE

- 頭過身就過：Loader、延遲載入
  - Process 剛啟動時的偵測比較多
  - 對抗 Sandbox、基於 Emulation 的偵測方法
- 對抗 / 繞過 User Space Hook
  - Direct Syscall
  - API Unhooking
- AMSI Bypass, Patch NTDLL!EtwEventWrite
  - .NET 與 PowerShell 的專屬偵測機制

Direct Syscall

DEVCORE

- EDR 會 Hook Windows API 來做動態偵測
  - 自己做 Syscall 可以跳過所有 User Space Hook
- Windows 更新版本會重新編號 Syscall Number
  - 檢查 Windows 版本後查表，缺點是新版 Windows 可能沒資料
  - 透過 NTDLL function 找到 Syscall Number
- 參考專案
  - github.com/klezVirus/SysWhispers3
  - github.com/MoePus/SPiCall <- 透過 KnownDlls Section 讀取 NTDLL

Plugin: Shellcode, .NET and BOF

DEVCORE

- 避免被分析或偵測，乾脆把所有功能做成 Plugin
  - Shellcode - github.com/TheWover/donut
  - Manual PE Loader - github.com/bb107/MemoryModulePP
  - Unmanaged .NET Loader - github.com/TheWover/donut
  - Cobalt Strike BOF (COFF Object) - github.com/trustedsec/COFFLoader

我們的 Insight

DEVCORE

- 以上招數組合後，傳統防毒軟體幾乎失效
- 唯一在靜態掃描階段就被偵測是 Golang 混淆工具導致的
- 有次被 EDR 阻擋的原因是 Command Line 參數
  - 改成從環境變數傳遞就繞過了

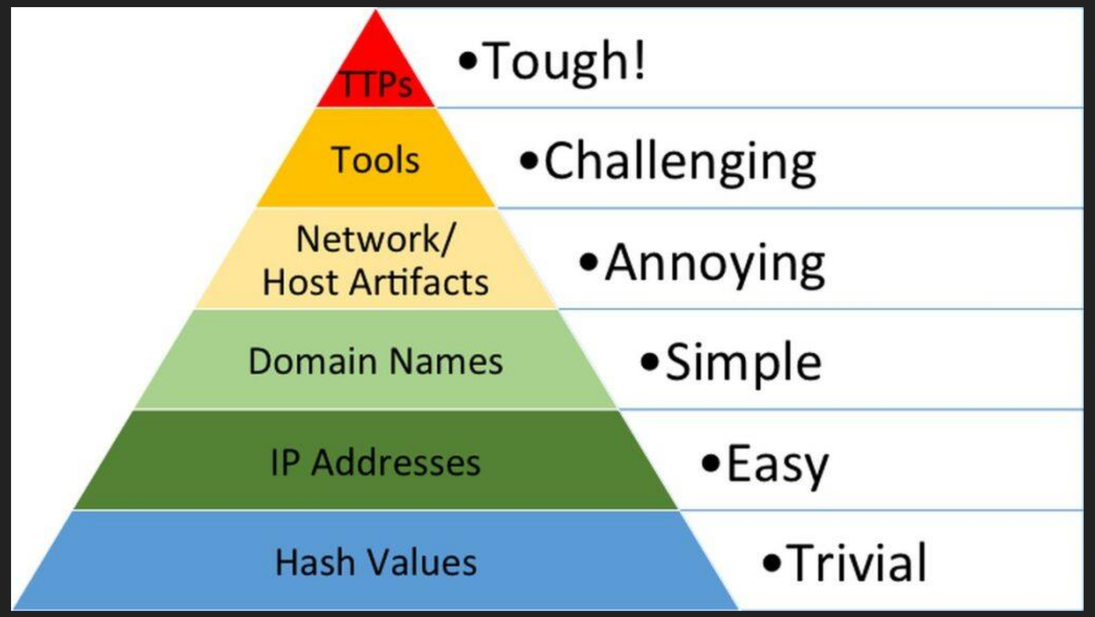
DEVCORE

攻擊者的風險

紅隊怕什麼？

DEVCORE

- 或者我們可以問要怎麼追蹤 APT 集團的活動？
- 慣用工具清單與專屬工具
  - 強化針對特定工具的偵測，逆向分析獨門工具建立特徵碼與偵測規則
- 發起攻擊的 IP 與 Domain
  - 防火牆規則封鎖或 Alert
- 常用攻擊手法、漏洞與習慣
  - 針對慣用手法與習慣建立偵測規則
  - 強化對應機器的監控與防禦



OPSEC：避免分析

DEVCORE

- 避免明文 Command line 參數傳遞
  - 曾經發生過參數出現 "--socks5" 字眼就被殺掉 QQ
- 減少建立新的 Process，能用 API 完成就別用外部指令
  - 減少依賴外部指令工具
- 移除 Command Line 使用說明與 Debug Log
  - 不然下個 -h /? 就知道程式有什麼功能

OPSEC：避免分析

DEVCORE

- 對投入使用的程式與 Payload 進行混淆
- 使用 Release 模式編譯並移除內嵌 Debug Symbol
  - 增加逆向分析成本
- 檢查產出 Payload 是否包含本機路徑、使用者名稱、內部網址、專案名稱
  - 會成為特徵，可以用來追蹤攻擊者
- 離開目標環境則改變自身行為，增加過期機制
  - 避免 Sandbox 自動化分析

軟體安全也是 OPSEC 的一環

DEVCORE

- 軟體供應鏈安全
  - 檢查第三方程式碼有無弱點、後門
  - 我們會進行全面 Code Review、維護內部 Fork
- 工具被竊取或重複使用
  - DRM 機制、到期失效機制、限制外部參數輸入能力
- 軟體品質與可靠度
  - 自動化測試、CI/CD
  - 從加上自動化測試後，問題數量~~客訴比例~~明顯下降

DEVCORE

背景知識與精進方向

背景知識與精進方向

DEVCORE

- 作業系統底層原理
  - 閱讀 GNU/Linux 程式碼，了解實作細節
  - 逆向 Windows DLL 與 Kernel，尋找更多攻擊面與繞過機會
- 程式語言特性
  - 學習各語言 Best Practice，減少繞路
- EDR/防毒軟體原理
  - 偵測技術、繞過方法

背景知識與精進方向

DEVCORE

- 新的攻防技術與知識
  - 新發現的攻擊面、Exploit Mitigation… 等
- 惡意程式分析技術與方法
  - 換個觀點會看到不一樣的世界
- 追蹤 X (Twitter) / GitHub 上其他人在做什麼
  - 社群平台推薦演算法會帶你看到更多相關東西

總結

DEVCORE

- 選擇適合的語言並善用其特性
- 了解防禦產品技術與攻防原理
- 混淆靜態內容並動態載入 Plugin，避免偵測與分析
- 持續自我精進，上 GitHub 長知識

DEVCORE

Thanks!

戴夫寇爾股份有限公司
contact@devco.re
02-2577-0925