MWS Cup 2017 課題 3 解説

MWS 2017 ポストミーティング 2017/11/28

課題3の概要

- マルウェア動的解析ログ(Cuckoo Sandbox の json ログ)の分析
  - [3-1] コードインジェクションを行う API コールの分析
    - インジェクション元とインジェクション先の PID およびプロセス名の特定
    - メモリ確保やコードの書き込みを行う API コールの特定、動作の説明
    - インジェクトしたコードを実行する API コールの特定、実行方式の説明
  - [3-2] Sandbox Fingerprinting と考えられる API コールの列挙
    - 挙動の概要、ログファイル名、PID, プロセス名、API コール
  - [3-3] 実践的な研究をするためにはどんなデータセットが必要か？

課題3の意図

- マルウェア動的解析ログ(Cuckoo Sandbox の json ログ)の分析
  - [3-1]
    - 過去問で予習してもらいたい
    - ノウハウを継承してもらいたい
  - [3-2]
    - マルウェアとサンドボックスの最新動向を調査してもらいたい
    - サンドボックスはどの程度作り込むべきか検討してもらいたい
  - [3-3]
    - MWSCup 参加者が研究用データセットについてどう考えているか知りたい

課題3-1 コードインジェクション分析の基本的なやり方

- コードインジェクションの流れとよく使われる API コールを探す
  1. 対象プロセス起動(ない場合もある)
     - ✓ **CreateProcessInternalW, NtCreateUserProcess**

  2. 対象プロセス内のメモリ確保およびコード書き込み
     - ✓ **NtAllocateVirtualMemory, NtProtectVirtualMemory, NtMapViewOfSection,**
     - ✓ **WriteProcessMemory, NtWriteVirtualMemory**

  3. 書き込んだコードの実行
     - ✓ **CreateRemoteThread, NtSetContextThread, NtQueueApcThread, NtResumeThread**

[Log1] WriteProcessMemory, CreateRemoteThread による典型的なインジェクション

1. 他のプロセスのメモリを操作をしている API コールを探す

- process_handle 0xffffffff は CurrentProcess (自プロセス)
- **[“arguments”][“process_handle”] != “0xffffffff”** を探す


```
3972, "C:¥Program Files¥Internet Explorer¥iexplore.exe"
NtAllocateVirtualMemory=>0, {"process_identifier"=>384, "region_size"=>4096,
"protection"=>64, "base_address"=>"0x00ba0000", "allocation_type"=>12288,
"process_handle"=>"0x000000fc"},

WriteProcessMemory=>1, {"buffer"=>"..", "base_address"=>"0x00ba0000",
"process_identifier"=>384, "process_handle"=>"0x000000fc"},

CreateRemoteThread=>172, {"thread_identifier"=>0, "process_identifier"=>384,
"function_address"=>"0x00ba0000", "flags"=>0, "stack_size"=>0,
"parameter"=>"0x00a00000", "process_handle"=>"0x000000fc"},
```

[Log2] NtMapViewOfSection で対象プロセスにメモリをマッピング、NtSetContextThread, NtResumeThread で実行

- コードの書き込みは周辺の *NtAllocateVirtualMemory*, *NtWriteVirtualMemory* の *section_handle* や *base_address* をトレースすることで解析可能


```
3548, "C:¥Users¥Smith¥AppData¥Local¥Temp¥7140e68ebe1(..)d3e8401e873bb937.exe" ,
CreateProcessInternalW=>1, {"thread_identifier"=>4024, "thread_handle"=>"0x000001a8",
"process_identifier"=>4020, "current_directory"=>"", "filepath"=>"", "track"=>1,
"command_line"=>"C:¥¥Windows¥¥system32¥¥svchost.exe", "filepath_r"=>"", "creation_flags"=>67108868,
"inherit_handles"=>0, "process_handle"=>"0x000001ac"},
NtMapViewOfSection=>0, {"section_handle"=>"0x00000128", "process_identifier"=>4020,
"commit_size"=>0, "win32_protect"=>64, "buffer"=>"", "base_address"=>"0x00200000",
"allocation_type"=>0, "section_offset"=>0, "view_size"=>122880, "process_handle"=>"0x000001ac"},
NtSetContextThread=>0, {"registers"=>{"eip"=>2228760, "esp"=>719852, "edi"=>0, "eax"=>2228224,
"ebp"=>719860, "edx"=>4399364, "ebx"=>2147348480, "esi"=>0, "ecx"=>0},
"thread_handle"=>"0x000001a8", "process_identifier"=>4020},
NtResumeThread=>0, {"thread_handle"=>"0x000001a8", "suspend_count"=>1,
"process_identifier"=>4020},
```

[Log3] *WriteProcessMemory, NtSetContextThread, NtResumeThread と NtMapViewOfSection, NtQueueApcThread, NtResumeThread の2パターン*

3292, "C:¥Users¥Smith¥AppData¥Local¥Temp¥cda34f6e5a38e873595698616 59(..)f9f4b50b719a3.exe" , CreateProcessInternalW=>1, {"thread_identifier"=>3432, "thread_handle"=>"0x000000a0", "process_identifier"=>3428, "current_directory"=>"", "filepath"=>"", .. "process_handle"=>"0x000000a4"},
NtSetContextThread=>0, {"registers"=>{"eip"=>2004906136, "esp"=>1245168, "edi"=>0, "eax"=>4274336, "ebp"=>0, "edx"=>0, "ebx"=>2147348480, "esi"=>0, "ecx"=>0}, "thread_handle"=>"0x000000a0", "process_identifier"=>3428},
NtResumeThread=>0, {"thread_handle"=>"0x000000a0", "suspend_count"=>1, "process_identifier"=>3428},
3428, "C:¥Users¥Smith¥AppData¥Local¥Temp¥cda34f6e5a38e873595698616 59e (..)f9f4b50b719a3.exe" , CreateProcessInternalW=>1, {"thread_identifier"=>3608, "thread_handle"=>"0x000000a4", "process_identifier"=>3604, "current_directory"=>"", "filepath"=>"", .. "process_handle"=>"0x000000a8"},
NtMapViewOfSection=>0, {"section_handle"=>"0x000000dc", "process_identifier"=>3604, "commit_size"=>159744, "win32_protect"=>64, "buffer"=>"", "base_address"=>"0x000a0000", "allocation_type"=>0, "section_offset"=>0, "view_size"=>159744, "process_handle"=>"0x000000a8"},
NtQueueApcThread=>0, {"function_address"=>"0x00000000", "parameter"=>"0x00000000", "thread_handle"=>"0x000000a4", "process_identifier"=>3604},
NtResumeThread=>0, {"thread_handle"=>"0x000000a4", "suspend_count"=>1, "process_identifier"=>3604},

課題3-1を解くスクリプトの例(ruby)

main.rb

```ruby
require 'json'

Dir.glob("**/*.json").each{|log|
  p log
  j = JSON.load(open(log))
  open(File.basename(log)+".csv", "w") {|f|
    f << "Category, APIName&RetVal, Arg, \n"
    j["behavior"]["processes"].each{|proc|
      print [proc["pid"], proc["command_line"], "\n"].join(", ")
        proc["calls"].each{|c|
          tag = nil
          if ["CreateProcessInternalW", "NtCreateUserProcess"].include?(c["api"])
            tag = "[PROC]"
          elsif ["NtAllocateVirtualMemory", "NtProtectVirtualMemory", "NtMapViewOfSection"].include?(c["api"]) && c["arguments"]["process_handle"] != "0xffffffff"
            tag = "[MEM]"
          elsif ["WriteProcessMemory", "NtWriteVirtualMemory"].include?(c["api"]) && c["arguments"]["process_handle"] != "0xffffffff"
            tag = "[WRITE]"
            c["arguments"]["buffer"] = ".."
          elsif ["CreateRemoteThread", "NtSetContextThread", "NtQueueApcThread", "NtResumeThread"].include?(c["api"])
            tag = "[EXEC]"
          end
          print [tag, c["api"] + "=>" + c["return_value"].to_s, c["arguments"], "\n"].join(", ") if tag
          f  << [tag, c["api"] + "=>" + c["return_value"].to_s, c["arguments"], "\n"].join(", ") if tag
      }
      p "-" * 100
    }
  }
}
```

課題3-2 Sandbox Fingerprinting の分析例

- マルウェアの耐解析機能で使われる API やパラメータ(リソース名等)がログに含まれるか調べる
- Cuckoo の Signature を参考に亜種パターンを探す
  - https://github.com/cuckoosandbox/community/tree/master/modules/signatures/windows
  - Signature のマッチング結果は json ログの “signatures” に出力、該当 API コールは ”marks”

- その他の手がかり
  - Black Hat 等の産業系セキュリティカンファレンス
  - 学術論文 (SandPrint: Fingerprinting Malware. Sandboxes to Provide Intelligence for Sandbox Evasion)
  - ツール
    - https://github.com/a0rtega/pafish
    - https://github.com/LordNoteworthy/al-khaser

配点・採点基準

課題		配点
3-1	3-1-1	**1**
	3-1-2	**1**
	3-1-3	**1**
	3-1-4	**1**
	3-1-5	**1**
	3-1-6	**1**
	3-1-7	**2**
	3-1-8	**2**
	3-1-9	**2**
3-2	3-2-1	**4**
	3-2-2	**4**
	3-2-3	**4**
3-3		**1**
	合計	**25**

- 3-1
  - 3-1-7～3-1-9は2件の正解があり、1件につき1点
  - 説明として書かれているパラメータが多少違っても、APIの呼び出しパターンがあっていれば正解
  - ログ中に存在しないAPIを書いているケースは明らかに間違っているので減点
- 3-2
  - CuckooのsignatureにCuckooに実装済みのパターンでも内容が概ね正しければおまけで2点
- 3-3
  - 何か書いていれば1点

課題作成・採点者

- 大山 恵弘 (筑波大学)
- 柴田 龍平 (NTT セキュリティジャパン株式会社)
- 森下 知哉 (NTT セキュリティジャパン株式会社)
- 村上 純一 (PwC サイバーサービス合同会社)
- 小林 鉄平 (株式会社 FFRI)
- 松木 隆宏 (株式会社 FFRI)