A7 SSH 查核點報告

1. 實作的 Activity

配合子計畫一誘導與干擾對手攻擊策略(Affect 影響目標)之相關工具開發，實作 Prevent Activity 與 Disrupt Activity。

Goals	Prepare	Expose		Affect			Elicit		Understand
Approaches	Plan	Collect	Detect	Prevent	Direct	Disrupt	Reassure	Motivate	Analyze
Activities	Cyber Threat Intelligence	API Monitoring	Introduced Vulnerabilities	Baseline	Attack Vector Migration	Isolation	Application Diversity	Application Diversity	After-Action Review
	Engagement Environment	Network Monitoring	Lures	Hardware Manipulation	Email Manipulation	Lures	Artifact Diversity	Artifact Diversity	Cyber Threat Intelligence
	Gating Criteria	Software Manipulation	Malware Detonation	Isolation	Introduced Vulnerabilities	Network Manipulation	Burn-In	Information Manipulation	Threat Model
	Operational Objective	System Activity Monitoring	Network Analysis	Network Manipulation	Lures	Software Manipulation	Email Manipulation	Introduced Vulnerabilities	
	Persona Creation			Security Controls	Malware Detonation		Information Manipulation	Malware Detonation	
	Storyboarding				Network Manipulation		Network Diversity	Network Diversity	
	Threat Model				Peripheral Management		Peripheral Management	Personas	
					Security Controls		Pocket Litter		
					Software Manipulation				

子計畫二將完成 2 種 Prevent 相關工具與 2 種 Disrupt 相關工具

	Expose		Affect			Elicit		Understand
	Collect	Detect	Prevent	Disrupt	Direct	Reassure	motivate	Analyze
子計畫一	2	2	2	2	4			
子計畫二			2	2		4	4	
子計畫三	2	2						1

選定實作的 activity 如下

Prevent	Disrupt
Baseline	Isolation
Hardware Manipulation	Lures
Isolation	Network Manipulation
Network Manipulation	Software Manipulation
Security Controls	

由於使用增強式學習(RL)，activity 會實作在 RL 的動作(action)裡

2. RL 的設計

RL 會設計成將回饋函式定義成將 MITRE Technique 轉為 Tactic 以辨認攻擊深度，再給予相關的獎勵值，期待 RL 的決策能夠達成增加攻擊深度（延長 kill chain 長度）的目標。

(1) RL 的狀態設計說明

$$s_t = \begin{cases} 0,\ if\ command\ is\ not\ mapped\ to\ any\ MITRE\ technique \\ the\ tactic\ index\ k, if\ command\ is\ mapped\ to\ a\ MITRE\ technique \\ and\ mapped\ to\ tactic\ k,\ k \in \{1,2,3,\cdots,14\} \end{cases}$$

為了希望 RL 能給出延長 kill chain 長度的決策，將 state 定義為攻擊指令對應到的 MITRE tactic 編號 k (由左到右 1~14)，如果沒有對到任何 technique/tactic 那就是 0。

(2) RL 的動作設計說明

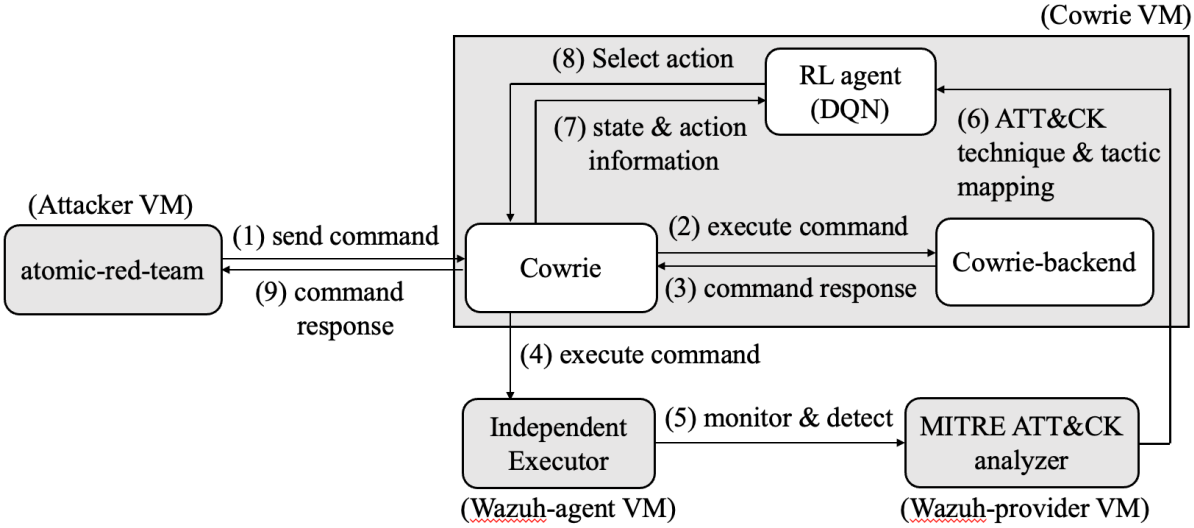
Action	Engage Activity					
	Baseline	Isolation	Network Manipulation	Security Controls	Lures	Software Manipulation
Restore the VM to original state	✓					
Kill long-running processes	✓					
Degrade network speed when attacker exfiltrates data			✓			
Add a user with a strong password				✓		
Change password of the user with a strong password				✓		
Add a user with a weak password					✓	
Change password of the user with a weak password					✓	
Block the access of network operation for another VM		✓	✓			
Allow the access of network operation for another VM			✓			

(3) RL 的反饋函式設計說明

$$r_t(s_t, a_t) = \begin{cases} -1, if\ session\ is\ terminated\ or\ connection\ is\ closed \\ (the\ tactic\ index\ k\ in\ s_{t+1}) \times W,\ W\ is\ a\ weight\ value \end{cases}$$

接著根據 state 的 tactic 編號值 k 給予 reward 值，若 tactic 越右邊則給越高的 reward 值，具體為 k 值乘上一個 weight 值 W (目的在於控制 reward 的增長速度)，W 目前暫時設為 0.5，也可以不乘或是乘別的數值，透過這樣的設計希望引導 RL 的決策能夠達成讓 kill chain 往越右邊的方向發展。

3. 系統架構與實作細節



整個系統是基於 Cowrie 的 open source 去做程式碼的修改與串接。首先從 atomic-red-team 搜集攻擊資料，安排攻擊進行 RL 的訓練，而偵測攻擊的部分會使用 Wazuh 這個開源工具，由於 Wazuh 具有偵測攻擊的 MITRE technique 功能，因此先用 Wazuh 偵測攻擊的 technique，接下來再用自己研發的方法將 technique 對應至 tactic。

在偵測攻擊時，由於 Wazuh 本身的用法，必須把要偵測的攻擊放在 Wazuh-agent 裡執行，讓 Wazuh-provider 來偵測，因此除了送到 Cowrie-backend 執行並回傳真實輸出外，還會再另外送到一個獨立的 VM 也就是 Wazuh-agent，這種方式同時也能夠避免被攻擊者發現有人在偵測他。

使用 python 撰寫系統串接程式，透過 socket 將攻擊指令送至 Wazuh-agent 去執行，接著使用 Wazuh 本身提供的 RESTful API 以及 Indexer 去撈 Wazuh-provider 偵測到的 MITRE technique，實作細節如下：
(附件提供程式檔 wazuh.py、cowrie.py、api.py)

Wazuh agent

(1) 將 wazuh.py 放入 Wazuh-agent 執行，他會監聽 9999 port，並執行收到的指令

(2) 將 cowrie.py 放入 cowrie 目錄底下執行，他會監控 /var/log/cowrie/cowrie.json 內容，如果有更動，會讀取新的部分，判斷 event 不是 owrie.command.input，如果是就會傳遞 input 給 wazuh.py

(3) 有些很長但不是 input 的 json 讀取會有錯誤產生，所以用 try 來忽略它

Wazuh api

(1) Wazuh api 會透過 9200 去跟 wazuh indexer 要 data
(2) data 要求的類型設為取得特定 ip 最新的 data
(3) 如需更改要求只需修改 data 的部分

DQN 的部分，使用 pytorch 函式庫實作，神經網路的模型架構如下，使用了簡單的線性層與 relu (relu 用來產生非線性變化)。

```
class DQN(nn.Module):
    def __init__(self, n_state, n_action):
        super(DQN, self).__init__()
        self.fc1 = nn.Linear(n_state, 256)
        self.fc2 = nn.Linear(256, 256)
        self.fc3 = nn.Linear(256, n_action)

    def forward(self, state):
        x = F.relu(self.fc1(state))
        x = F.relu(self.fc2(x))
        return self.fc3(x)
```

由於目前 RL 訓練尚未完全完成，還需要再做調整，因此暫時無法提供完整程式碼或實作成果。

4. MITRE technique 對應至 tactic 之方法

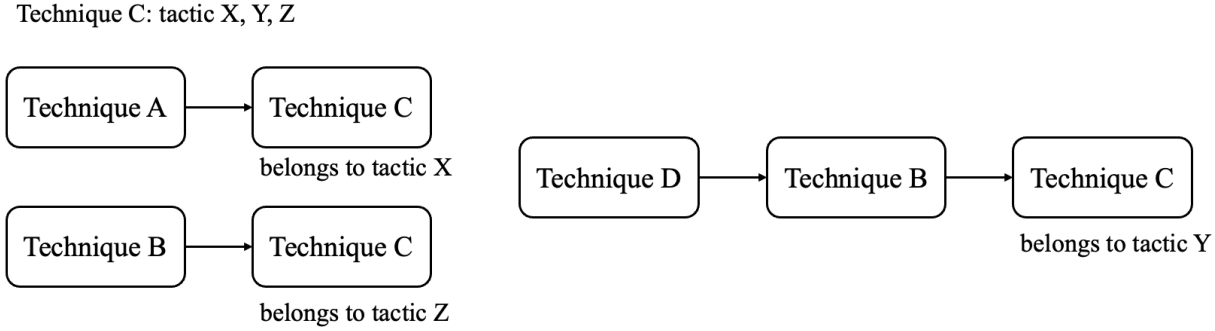
前面講到已先用 Wazuh 將攻擊指令轉換為 technique，而全部的 200 多個 technique 中，絕大多數的 technique 只會對到唯一的 tactic，只有其中的 27 個 technique 有可能會對到不同的 tactic，因此只需要做這部分，需要做的 27 個 technique 如下

(1) Abuse Elevation Control Mechanism
(2) Access Token Manipulation
(3) Account Manipulation
(4) Adversary-in-the-Middle
(5) BITS Jobs
(6) Boot or Logon Autostart Execution
(7) Boot or Logon Initialization Scripts
(8) Content Injection
(9) Create or Modify System Process
(10) Debugger Evasion
(11) Deploy Container
(12) Domain Policy Modification

(13)Event Triggered Execution
(14)External Remote Services
(15)Hijack Execution Flow
(16)Input Capture
(17)Modify Authentication Process
(18)Network Sniffing
(19)Pre-OS Boot
(20)Process Injection
(21)Replication Through Removable Media
(22)Scheduled Task/Job
(23)Software Deployment Tools
(24)Traffic Signaling
(25)Use Alternate Authentication Material
(26)Valid Accounts
(27)Virtualization/Sandbox Evasion

我們的對應方法會是利用 state machine 的方式，透過在 atomic-red-team 蒐集夠多的攻擊資料(已有 technique 和 tactic 的 label)來進行前後關係的判斷，最後會寫成自動化的規則判斷腳本。

做法為當一個 technique 進來的時候，先去觀察目前所屬的 technique/tactic 狀態，透過目前所處的狀態來決定進來的 technique 屬於哪個 tactic。



例如 technique C 可能會對到 tactic x,y,z，當 technique C 進來的時候如果目前狀態是 technique A，由於在搜集的 label data 裡有找到這樣的情況會是 label 為 tactic x，因此判斷他會對到 tactic x。當 technique C 進來的時候如果目前狀態是 technique B，由於在搜集的 label data 裡有找到這種情況會是 label 為 tactic z，因此判斷他是對應到 tactic z。

不過也有可能單純這樣做會判斷不出來，此時會去多看前一個 technique/tactic 狀態，再去看搜集的資料裡有沒有此情形，如果有找到這種情況並且 label 成 tactic Y 的話，那就判斷他是對到 tactic Y，不過這邊只是觀念上大概的做法而已，實際上在判斷時會更加複雜。