### Motivational Example

### 

### Graph Definition (𝑁 , 𝐸, 𝑉 , 𝑃)

Node (N): access level (have a foothold/a privileged account[/persistence] on machine x),

Edge (E): foothold, privilege escalation, data exfiltration (and maybe persistence)

Value (V): value of exfiltrated data

Probability (P):

*Model as if everything is exploitable given enough time:*

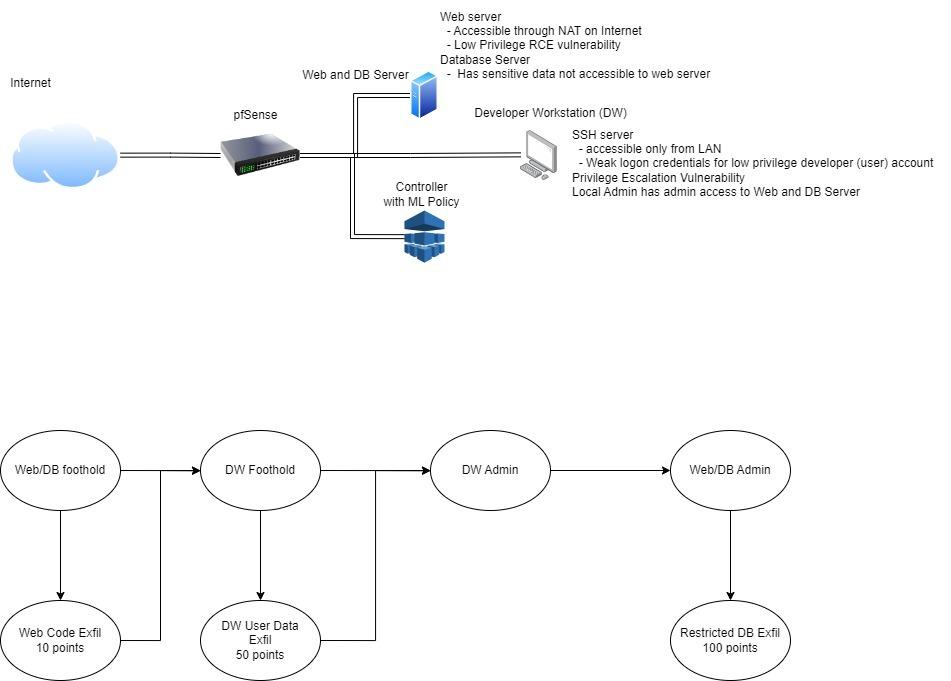
Apparent Time Cost- how vulnerable something seems (known vulnerabilities and their CVSS score)

True Time Cost - Probabilistic distribution describing how long the exploit would take a given hacker

Probability - inverse of time

Neither the attacker nor the defender know the true probability. They must both use available data and heuristics to reason about how vulnerable something is. E.g., vulnerability scans and risk analysis of running services (how risky is it to have an SSH server running even if it’s up to date?). We can use a vulnerability and network scanner with a few hard coded heuristics to give the model its perception of the probabilities associated with each edge.

If the defender makes a fake edge, of course, it will be aware that it is not a viable attack path.



### Attacker’s Action Space (Aa)

| Abstract Action on the Attack Graph | Potential Implementations | Example |
| --- | --- | --- |
| Traverse edge | Gain foothold | Gets a low priv shell |
|  | privesc | Gets a high priv shell |
|  | exfil | Gets data off machine |
|  | (establish persistence?) | Installs control/spy software |
| Stay in same node | Do nothing |  |

### Defender’s Action Space (Ad)

| Abstract Action on the Attack Graph | Potential Implementations (Defensive or Deceptive Techniques) | Example |
| --- | --- | --- |
|  | Deploy honeypot 🕵️ |  |
|  | Start honey service 🕵️ |  |
|  | Mix up network addresses 🕵️ |  |
|  | Reset password 🛡️ |  |
|  | Create fake data 🕵️ |  |
|  | Block IP 🛡️ |  |

### Attacker’s Observation (Oa)

| Abstract Information on the Attack Graph | Potential Implementations (Information about the network/defender and reconnaissance techniques to collect them) | Example |
| --- | --- | --- |
| Attacker Visible edges | 🦶Open ports (port scan) |  |
|  | 🦶Service detection (banner grab; manual interaction) |  |
|  | 🦶Network hosts (host scan) |  |
|  | ⬆️/📨 Files on host (ls) |  |
|  | ⬆️ Running processes (ps) |  |
|  | ⬆️ System configs (various) |  |

### Defender’s Observation (Od)

| Abstract Information on the Attack Graph | Potential Implementations (Information about the network/attacker and telemetry techniques to collect them) | Example |
| --- | --- | --- |
| Vision threshold of attacker | Collected alerts from IDS and EDR systems. | Threshold increases when data is collected |
| Graph of where the defender sees the attacker | Attacker actions | Defender keeps a graph of the different nodes he has been able to see the attacker in so that it can analyze it later and gather information |
| Transition Probabilities | Vulnerability + Network Scans -> Heuristic engine that we build -> |  |

### Reward Scheme for Attacker (Ra)

(Potential objectives of attackers in the real world? Cost?)

Objectives:

1. Degrade system
2. Steal data
3. Modify data
4. Establish control

Cost:

1. Revealing information about identity and methods

We could focus on

Objective: Exfil data without revealing info about self

### Reward Scheme for Defender (Rd)

(Potential objectives of defenders in the real world? Cost?)

We could focus on

Objective: Gather threat intel without losing data

Objectives:

1. Find attacker in the network
2. Gather information about attacker (IP, how they broke into the network, damage performed)
3. Kick attacker out of the network
4. Trap attacker through deceptive events

Cost:

1. Potential damage caused by the attacker
2. Actions needed to defend against attacker could compromise availability/functionality/performance of the network and its assets (ex: needing to redirect traffic and overloading the data pipeline with all the traffic)
3. Time/productivity from having to defend

Significant Implementation Components

* Defender Action (we can start with a small subset of all possible actions we could imagine)
  + Deploy honeypot
  + Start honey service
  + Create tracked files on hosts
* Infrastructure
  + Build and configure network
* Defender Perception
* IDS/EDR for live alerts about attacker
  + Or is it better if we only use observations from our traps?
* Heuristic vulnerability engine(Vulnerability + Network Scans)
  + In the short term, we could just manually define this information for the defender as it will be static

Adam’s reflections

***Probabilistic Position***

If we use traps as our only means of identifying our target, it will significantly simplify our implementation and will focus the game on the cyber deception techniques.

Once an attacker sets off a trap, we have them. We have 100% confidence that they are in a specific stage of their attack (or node on the graph). After time continues, we become less sure of where they are, but we can make a probabilistic prediction based on their last known prediction and the elapsed time.

***Defender Goals and Constraints***

The defender wants to learn about the attacker and not be attacked (whereas the hacker wants secrecy and to attack). The defender should try to locate the attacker and either 1) block it from the system or 2) lure it into exposing itself without causing damage.

Once we’ve identified the attacker even once, we can kick it out at any time, even if we lose track of it. After initial detection, we should try to learn as much as we can about the attacker before kicking it out (while balancing this against the risk of letting it stay in the system). We can accomplish this by strategically, covertly luring the attacker into interacting with more honey (pots, nets, services, etc).

In the previous model, we constrain the defender with a budget. While this could still be applied, it may be sufficient to have the defender constrained by a desire to have its actions go undetected. The defender is welcome to flood the network with honey and this *may* allow it to quickly detect the attacker; however, it will likely lose the advantage of having its tactics go undetected. If it can be stealthy, it may have better odds of keeping the attacker and gently leading it to the right place.

Too much honey may scare the attacker away, which is better than being attacked, but less good than safely getting threat intelligence.

***Should the defender defend?***

In order to focus on the deception, one option would be to have the end of game be banning an attacker’s IP. We could then use banning as the only defense mechanism and not worry about an attacker getting a new IP and carrying out the same attack.

Even if we choose to make banning the end of game, we could still use defense mechanisms as a way to steer the attacker where we want it to go. This seems interesting, but would certainly add to the complexity. Maybe that could be in Version 2?

***Computing Scores***

*Data exfil -* we can associate value with each piece of data, likely asymmetrically; e.g., losing info may cost the company a lot more than the data is worth to the attacker

*Threat intel -* there are many ways we could compute this. We could use simple heuristics like 1) number of interactions with traps or 2) area under curve of location certainty function over time. We could also do more complicated things like measuring what information they actually disclosed (malware installed \* how novel it is; passwords they tried in a brute force; privesc attempts; etc). This would be very interesting but might be another one that would be better in a Version 2. Again, this could be an asymmetric cost/reward.

**Model:**

**Attacker:**

* Every machine is represented in the graph by different nodes each one representing the different privilege escalation available in the node. The neighboring nodes are those you can access from the previous state, including those from other hosts.
* We assume we always start from the lowermost privilege node in the gateway if the network
* Action space: all the neighboring non-compromised nodes from attacker’s current state
  + Represented by a vector of 0s and 1s where 1 represents the neighboring nodes
  + Action: represented by a vector of -1, 0, 1, where -1 is the node the attacker is leaving from, 1 is the one it’s going to, and 0 are the rest.
* State: a graph of all compromised nodes
  + Represented by a graph of -1, 0, 1 where -1 are all the compromised nodes, 1 are the neighboring nodes, 0 are the ones we don’t have access to

**Defender:**

* Same as in the paper

Goal:

* Deploy the current model on the aws network.
* Automate network deployment with terraform
* Open the network and test it against outside attackers