Automated Adversary Emulation: A Case for Planning and Acting with Unknowns

Doug Miller, Ron Alford, Andy Applebaum, Henry Foster, Caleb Little, and Blake Strom
The MITRE Corporation
7515 Colshire Drive
McLean, Virginia 22102
{dpmiller, ralford, aapplebaum, hfoster, clittle, bstrom}@mitre.org

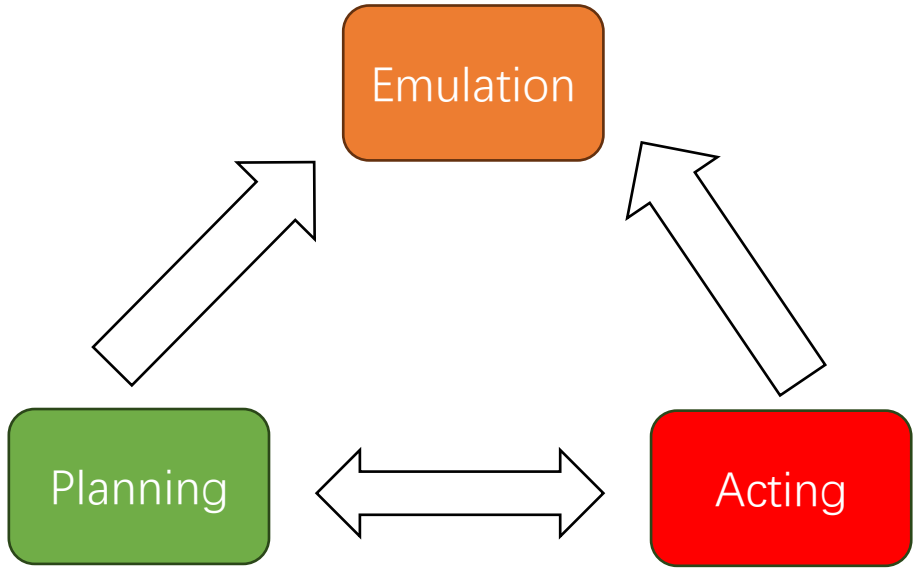
Reporter: Yangyang Wei

Outline

- Problem Statement
- Building Automated Adversary Emulation
  - Adversary Model
- Characterizing Uncertainty in Automated Adversary Emulation
- Formal Problem Description
- Representing Planning Problems for Adversary Emulation
- Choosing Adversary Techniques
- Discussion

Problem Statement

- Previous research works:
  - Automated penetration testing
    ⇔
    Automated adversary emulation
  - A strictly planning problem



- The work of this paper :
  - Automated adversary emulation
    ⇔
    A joint planning and acting problem
  - Real adversaries work in the face of unbounded uncertainty
  - Adversaries are unable to enumerate the outcomes of a sensing action without actually executing it.

Building Automated Adversary Emulation

Specific goals driving automated adversary emulation include:

1. Intelligent
   - The system should choose and chain actions in ways similar to how an adversary would.

2. Low Overhead
   - Defenders should be able to use the tool without needing explicit configuration details.

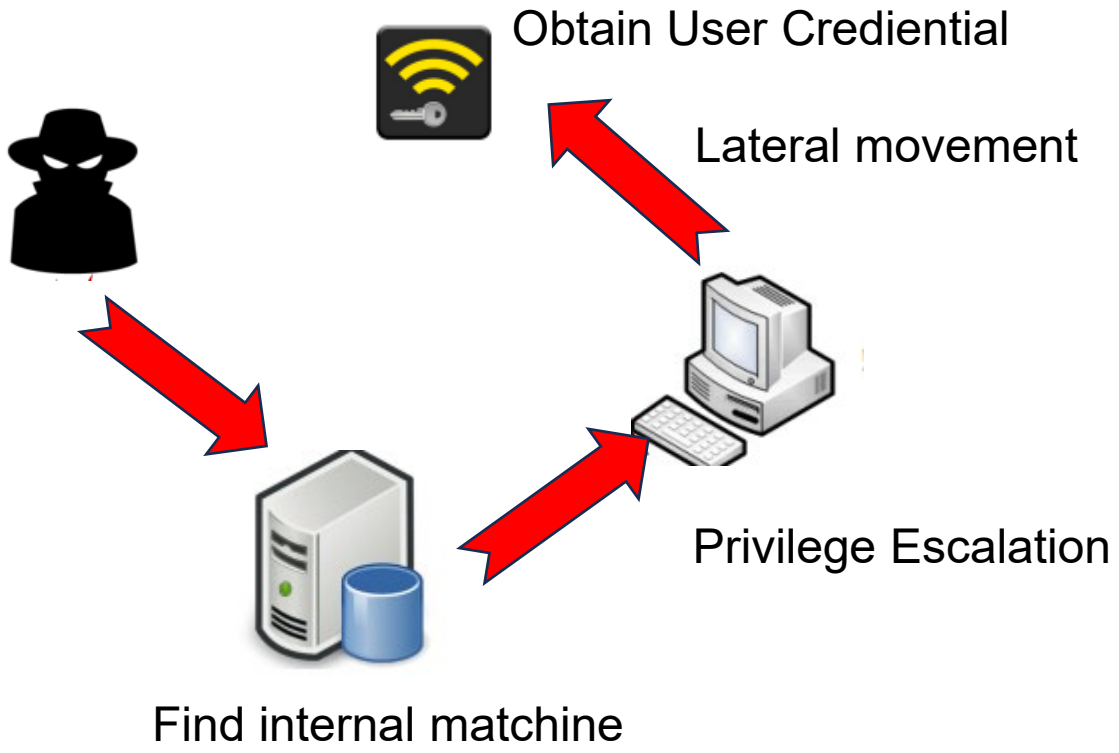
3. Realism
   - The system should execute the same techniques that a real adversary would, and, like a real adversary.

4. Modular
   - Users of the system should be able to run assessments with techniques of their choosing, as well as have the ability to add new techniques.

Adversary Model

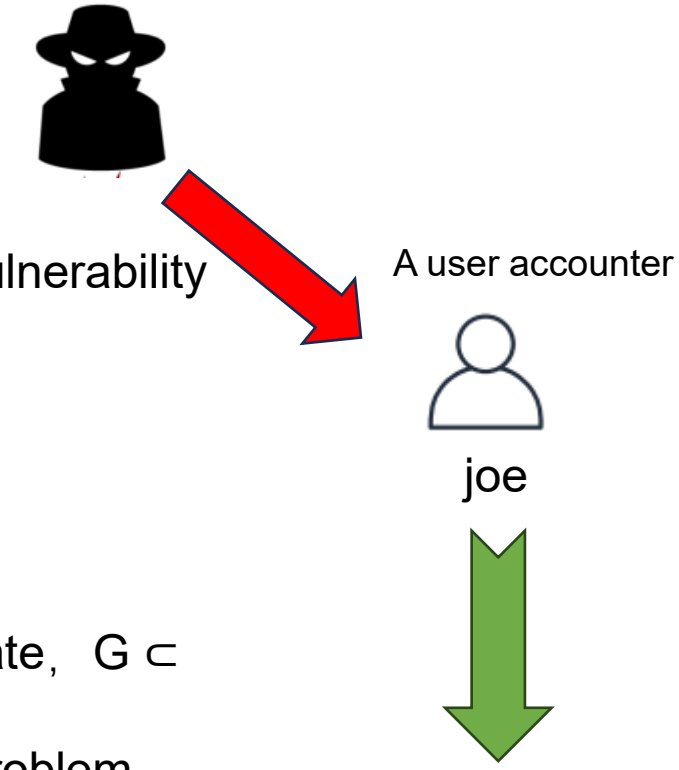
Characterizing Uncertainty in Automated Adversary Emulation

- Tree common actions
  - adversary typically execute during engagements
    - Exploiting vulnerability
      - Find a remote system and a kernel vulnerability
    - Remote System Discovery
      - Methods: ping, phish, etc.
    - Credential Dumping
      - Include: passwords, hashed passwords, etc.



Possible Uncertainty (each of these techniques in vastly different ways.)

- Exploiting vulnerability:
  - Is the target susceptible to this exploit
  - Was the exploit technique executed successfully
- Credential Dumping
  - no credentials;
  - credentials for accounts it has never heard of
  - credentials that it can not currently use;
  - credentials that it can immediately use.



Formal Problem Description

- the large, agent-agnostic description of the problem ($\pi = \langle P, A, I, G \rangle$)
  1. P is a set of propositions， A is a set of actions， $I \subset P$ is starting state， $G \subset P$ is goal proposition.
  2. π is a sequence of actions $S = \{a1, a2, ..., an\} \Leftrightarrow$ A solution of the problem
- the smaller adversary-oriented view of the problem ($\pi i = <Fi, A, I, G>$
  1. $Fi \subseteq P$ is set of proposition at time i.

Representing Planning Problems for Adversary Emulation

Principles:

- Object-oriented description.

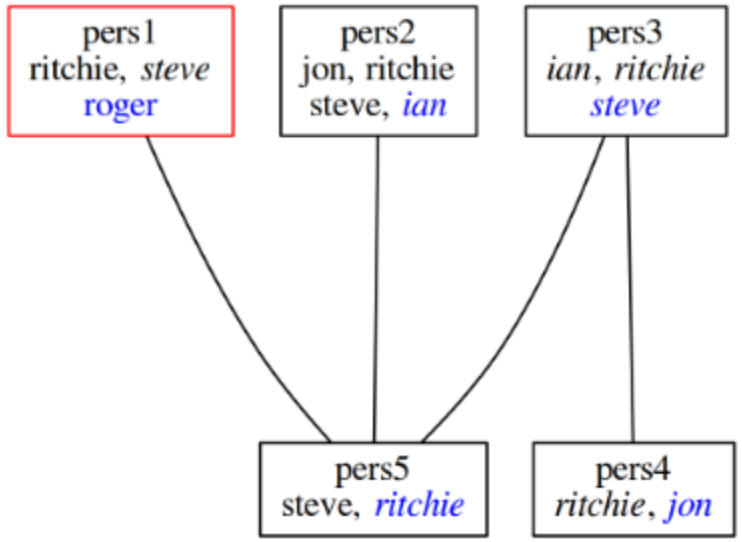
  the objects (i.e., hosts, users, accounts, etc.) typically found in networks

- Parameterized actions.

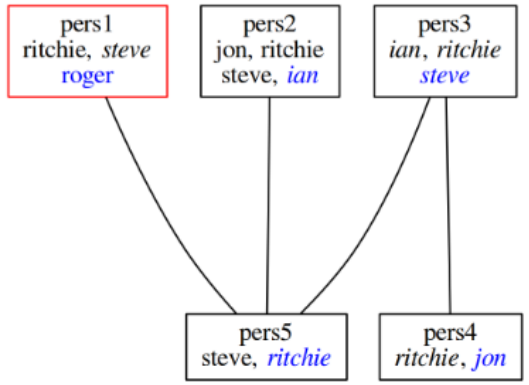
  refer to our action space as a set of ungrounded, parameterized actions

- Deterministic action outcome.

- Monotonic action consequences



Early Work in K



The first predicate connected(X, Y) denotes that two hosts in the model can communicate over the network

The second adds it to the adversary's knowledge base.

```
connected(X, Y) requires host(X), host(Y).
knowsConnected(X, Y) requires host(X), host(Y).
```

execute the enumerateHost action, it must have an escalated foothold (i.e., executing under root or SYSTEM) on a host

```
executable enumerateHost(X) if hasFoothold(X),
   escalated(X), not hostEnumerated(X).
caused knowsConnected(X, Y) if connected(X, Y)
   after enumerateHost(X).
caused hostEnumerated(X) after enumerateHost(X).
```

the adversary can dump credentials: it only needs an escalated foothold on a host to do so

```
executable dumpCreds(X) if hasFoothold(X), escalated(X).
caused knowsCreds(A) if activeCreds(A, X)
     after dumpCreds(X).
```

Pythonic Representation

Limitation: K did not follow the object-oriented guideline.

- preconditions, which must be true to execute the technique.
- postconditions, which at least will be true after executing the technique
- pre-properties, which are things that must be defined to execute the technique
- post-properties, which are things that will be defined after executing the technique

```
preconditions = [("rat", OPRat),
  ("share", OPShare({"src_host": OPVar("rat.host")}))]
postconditions =
  [("file_g", OPFile({'host': OPVar("share.dest_host")}))]
preproperties = ['rat.executable', 'share.share_path']
postproperties = ['file_g.path']
```

```
Parameters:
      EXECUTABLE, HOST, RAT, SHARE, SHARE_PATH, SRC_HOST
Preconditions:
      has_property(RAT, executable, EXECUTABLE)
      has_property(RAT, host, SRC_HOST)
      has_property(SHARE, dest_host, HOST)
      has_property(SHARE, share_path, SHARE_PATH)
      has_property(SHARE, src_host, SRC_HOST)
      oprat(RAT)
      opshare(SHARE)
Postconditions:
    + defines_property(FILE_G, path)
    + has_property(FILE_G, host, HOST)
    + opfile(FILE_G)
```

Choosing Adversary Techniques

- planningand-acting paradigm
  1. Obtain and update the world state.
  2. With the world state, use the precondition model to identify which actions are valid at the current time step.
  3. Building off of step construct a set of plausible plans.
  4. Evaluate each plan constructed in the previous step.
  5. Execute the first action in the highest rated plan.
  6. Observe the responses, stopping if the goal state has been observed and going back to step 1 otherwise.

Constructing Plans

1. Construct a fictional world P′.

2. Merge P′ with Fi. In the case that some proposition in P′ conflicts with Fi, defer to the known proposition to ensure consistency.
3. Initialize an empty set of plans, P .

4. For each action type – i.e., for each ungrounded action obtain the set of plans that executes that action the soonest. For example, if we can dump credentials – regardless of the host – at time step three and no sooner, add all plans of length three that dump credentials to P . Repeat this for each action type.

5. Return P .

$$S(p) = \sum_{i=1}^{n} \frac{R(a_i)}{i}$$

```
Parameters:
        RAT
Preconditions:
        oprat(RAT)
Postconditions:
      + defines_property(HOST_G, fqdn)
      + defines_property(HOST_G, os_version)
      + ophost(HOST_G)
      + oposversion(OS_VERSION_G)
```

Discussion

- Q1:As the investigation deepens, the number of potential solutions to enumerate increases significantly, resulting in a large number of possible combinations.

- Q2:How is this solution applied in real-world scenarios?