Decomposing the Network to perform Attack Planning under Uncertainty

Carlos Sarraute

CoreLabs & ITBA PhD program
Buenos Aires, Argentina

Hackito Ergo Sum – April 12-14, 2012

Presentation

One foot in the industry → Researcher in CoreLabs

12 years of experience in Information Security.
Some areas of interest:

- Vulnerability research
  - Bugweek
  - Publication of advisories
- Cyber-attack planning and simulation
- Improving OS detection using neural networks

One foot in the academy

M.Sc. in Pure Mathematics (UBA)
Finishing a Ph.D. in Informatics Engineering (ITBA)

- Director: Gera (a.k.a. Gerardo Richarte)

Agenda outline

1. Motivation
2. On Exploit Quality Metrics
3. The Search for an Efficient Solution
   - Planning for dummies
   - Two primitives
   - Using the primitives in a Network Graph
   - Integration with a Pentesting Tool
4. The Search for a Better Model
   - POMDPs
   - Penetration Testing as POMDPs
   - Decomposition in 4 Abstraction Levels
   - Experiments
5. Conclusion

Agenda

What is Penetration Testing?

Penetration testing

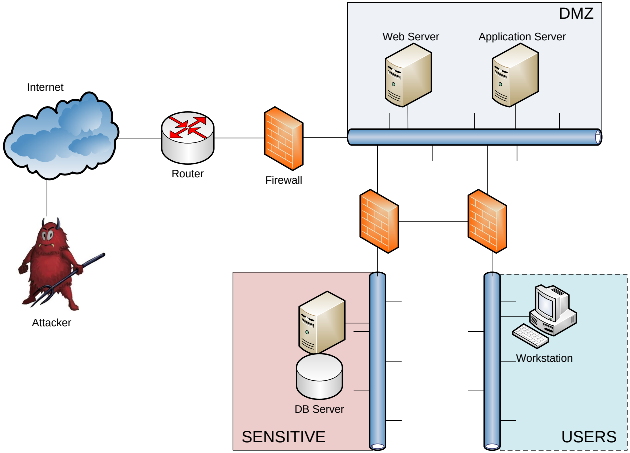
Actively verifying network defenses by conducting an intrusion in the same way an attacker would.

- Penetration testing tools have the ability to launch real exploits for vulnerabilities.
  - different from vulnerability scanners (Nessus, Retina, ...)
  - **no false positives!**
- Main tools available:
  - Core Impact (since 2001)
  - Immunity Canvas (since 2002)
  - Metasploit (since 2003)

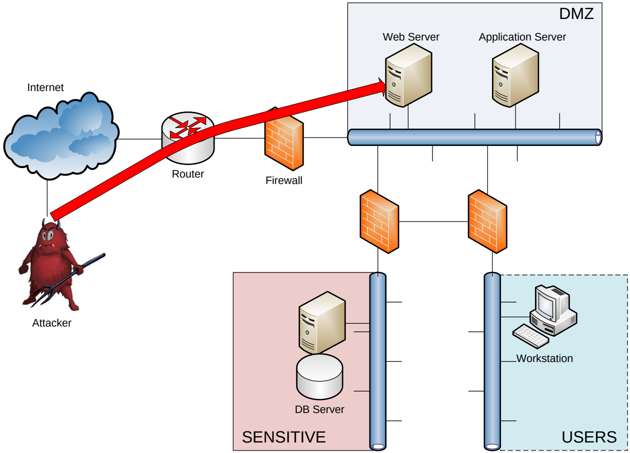
Need for Automation

- Reduce human labor
- Increase testing coverage
  - Higher testing frequency
  - Broader tests trying more possibilities
- Complexity of penetration testing tools
  - More exploits
  - New attack vectors (Client-Side, WiFi, WebApps, . . .)
- Equip penetration testing tool with “expert knowledge”
- Construct attack plans that pivot.

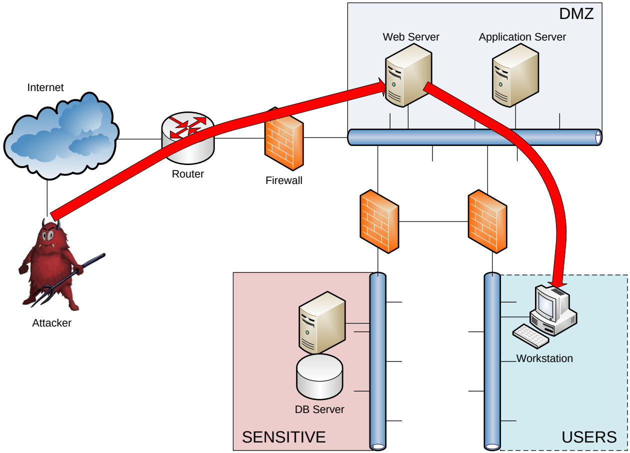
Anatomy of a real-world attack



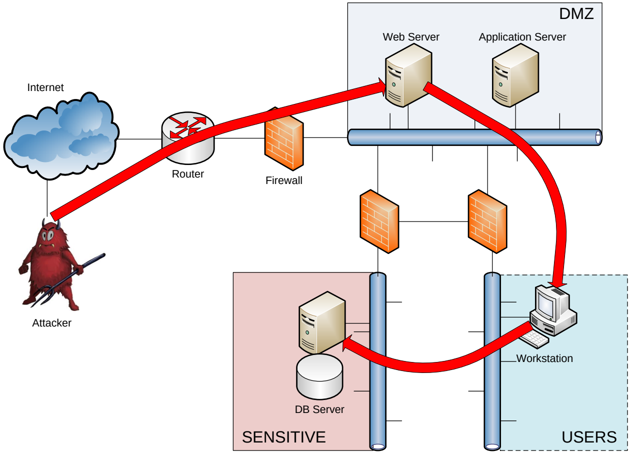
Anatomy of a real-world attack



Anatomy of a real-world attack



Anatomy of a real-world attack



Agenda

1. Motivation
2. On Exploit Quality Metrics
3. The Search for an Efficient Solution
   - Planning for dummies
   - Two primitives
   - Using the primitives in a Network Graph
   - Integration with a Pentesting Tool
4. The Search for a Better Model
   - POMDPs
   - Penetration Testing as POMDPs
   - Decomposition in 4 Abstraction Levels
   - Experiments
5. Conclusion

Basic definitions (see [Arc05])

Vulnerability (noun) A flaw in a system that, if leveraged by an attacker, can potentially impact the security of said system

- Also: security bug, security flaw, security hole

Exploit (verb) To use or manipulate to one's advantage (Webster)

Proof of Concept exploit - PoC (noun) A software program or tool that exploits a vulnerability with the sole purpose of proving its existence.

Exploit Code (noun) A software program or tool developed to exploit a vulnerability in order to accomplish a specific goal.

- Possible goals: denial of service, arbitrary execution of code, etc

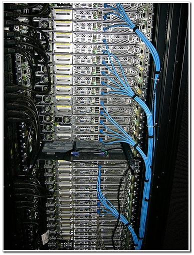
What can we measure? (I)

- Average running time
  - Straightforward to measure.
  - Some exploits require brute forcing $\longrightarrow$ sometimes that can be upgraded to more clever techniques
- Success rate or Probability of success
  - Success rate of testing an exploit repeatedly against a given platform.
  - Approximate different capacities, such as resilience to machine load, network load, or different configurations.
- Network traffic generated
- User required interaction
  - Determining if the exploitation of a bug will be "interactive" or unattended is an important piece of documentation.

What can we measure? (II)

- Targets exploited / known vulnerable targets
  - A vulnerability affects a set of platforms, for example, Windows XP SP2 and SP3 can be affected.
  - Variations in libraries in intra-service-pack patches or when different languages are supported may affect the exploit.
- Resilience to changes in configuration and machine load
  - Exploit for a vuln may only work with the default configuration.
  - Exploit use methods (such as hardcoded address) that are sensitive to minor changes in memory layout.
  - Exploits are more reliable when non-default configurations are used during development, and when they are tested in real-life use conditions.

How do we measure those values?



1. Use the Exploit Testing team infrastructure.
   - 748 virtual machines with different OS and applications.
   - Automated execution of all the exploits against vulnerable images... every night!
   - Statistics are extracted from the database of executions.
2. Get feedback from users.
   - Anonymized feedback program in Core Impact.

Agenda

Agenda

Simple brain teaser

In which order would you execute these exploits?

An obvious problem

Action	*Time*	*Probability*
$Exploit_1$	$8s$	$0,85$
$Exploit_2$	$100s$	$0,05$

Simple brain teaser

In which order would you execute these exploits?

An obvious problem

Action	*Time*	*Probability*
$Exploit_1$	$8s$	$0,85$
$Exploit_2$	$100s$	$0,05$

And maybe not so obvious

Action	*Time*	*Probability*
$Exploit_1$	$8s$	$0,05$
$Exploit_2$	$100s$	$0,85$

Solution

We suppose the actions are independent, so the expected total running times are:

$$t_1 + (1 - p_1) \cdot t_2 <^? t_2 + (1 - p_2) \cdot t_1$$

Solution

We suppose the actions are independent, so the expected total running times are:

$$t_1 + (1 - p_1) \cdot t_2 <^? t_2 + (1 - p_2) \cdot t_1$$

$$t_1 + t_2 - p_1 \cdot t_2 <^? t_2 + t_1 - p_2 \cdot t_1$$

Solution

We suppose the actions are independent, so the expected total running times are:

$$t_1 + (1 - p_1) \cdot t_2 <^? t_2 + (1 - p_2) \cdot t_1$$

$$t_1 + t_2 - p_1 \cdot t_2 <^? t_2 + t_1 - p_2 \cdot t_1$$

$$p_2 \cdot t_1 <^? p_1 \cdot t_2$$

Solution

We suppose the actions are independent, so the expected total running times are:

$$t_1 + (1 - p_1) \cdot t_2 <^? t_2 + (1 - p_2) \cdot t_1$$

$$t_1 + t_2 - p_1 \cdot t_2 <^? t_2 + t_1 - p_2 \cdot t_1$$

$$p_2 \cdot t_1 <^? p_1 \cdot t_2$$

$$\frac{t_1}{p_1} <^? \frac{t_2}{p_2}$$

Solution and second brain teaser

Best order

Action	*Time*	*Probability*	t/p
$Exploit_1$	$8s$	$0,05$	160
$Exploit_2$	$100s$	$0,85$	$117,6$

Solution and second brain teaser

Best order

Action	*Time*	*Probability*	t/p
$Exploit_1$	$8s$	$0,05$	160
$Exploit_2$	$100s$	$0,85$	$117,6$

What happens with more?

Action	*Time*	*Probability*
$Exploit_1$	$8s$	$0,05$
$Exploit_2$	$100s$	$0,85$
$Exploit_3$	$40s$	$0,50$
$Exploit_4$	$2s$	$0,01$

Solution and second brain teaser

Best order

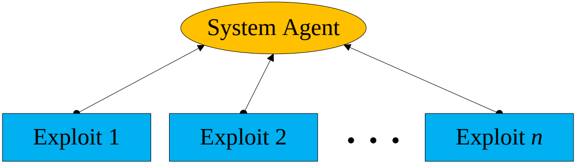
Action	*Time*	*Probability*	t/p
$Exploit_1$	$8s$	$0,05$	160
$Exploit_2$	$100s$	$0,85$	$117,6$

What happens with more?

Action	*Time*	*Probability*	t/p	*Order*
$Exploit_1$	$8s$	$0,05$	160	3
$Exploit_2$	$100s$	$0,85$	$117,6$	2
$Exploit_3$	$40s$	$0,50$	80	1
$Exploit_4$	$2s$	$0,01$	200	4

Agenda

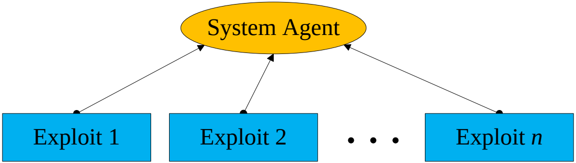
The **Choose** primitive



Problem

$\{A_1, \ldots, A_n\}$ independent actions that result in a goal $\mathfrak{g}$.
Each A_k has probability of success p_k and running time t_k.
Task: Find order of execution to minimize total running time.

The **Choose** primitive



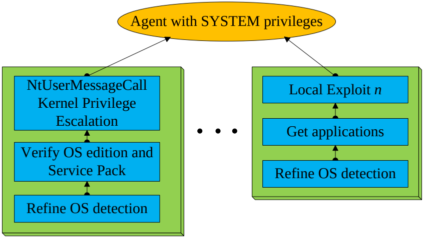
Problem

$\{A_1, \ldots, A_n\}$ independent actions that result in a goal $\mathfrak{g}$.
Each A_k has probability of success p_k and running time t_k.
Task: Find order of execution to minimize total running time.

Solution

Order actions according to t_k / p_k (in increasing order).

The **Combine** primitive



Definition

We call *strategy* a group of actions that are executed in a fixed order.

Problem

$\{G_1, \ldots, G_n\}$ are strategies that result in a goal $\mathfrak{g}$.
Task: Minimize total time.

Expected probability and time

If the actions of G are $\{A_1, \ldots, A_n\}$ then:
The expected running time of G is

$$T_G = t_1 + p_1\, t_2 + p_1\, p_2\, t_3 + \ldots + p_1\, p_2 \ldots p_{n-1}\, t_n$$

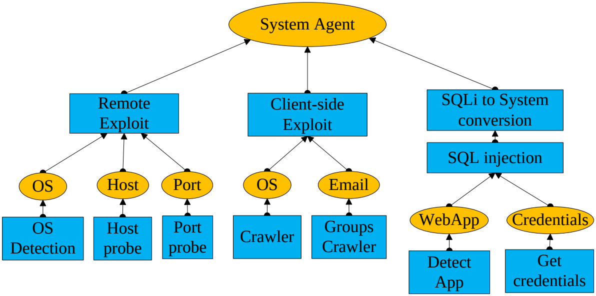
The probability of success is simply

$$P_G = p_1\, p_2 \ldots p_n$$

Solution

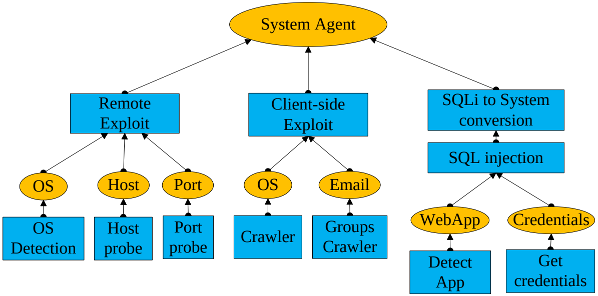
Sort the strategies according to T_G / P_G.
In each group, execute actions until one fails or all the actions are successful.
Complexity of planning: $\mathcal{O}(n \log n)$

The **Combine** primitive (cont)



Groups of actions with an AND relation (order is not specified).

The **Combine** primitive (cont)



Groups of actions with an AND relation (order is not specified).

Idea

In each group, order actions according to $t_k/(1 - p_k)$.

Intuitively, actions with higher probability of failure have priority.

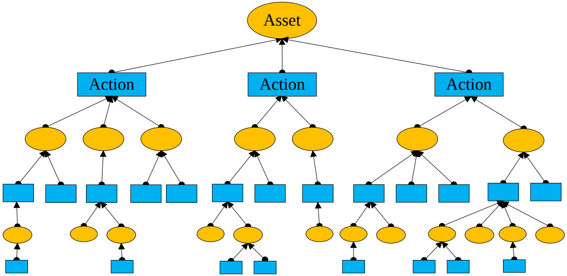
References (for this section)

- [Sar09a] New Algorithms for Attack Planning
  - *FRHACK Conference, France. Sept 7/8, 2009.*
- [Sar09b] Probabilistic Attack Planning in Network + WebApps Scenarios
  - *H2HC Conference, Sao Paulo, Brazil. Nov 28/29, 2009.*

Agenda

First level: fixed source and target

Given a source machine and a target machine, the problem is to find a path in an Attack Tree:



1. *Action node:* connected by AND relation with its requirements ⟶ use *Combine* primitive.
2. *Asset node:* connected by OR relation with the actions that provide that asset ⟶ use *Choose* primitive.

Second level: graph of machines

Use First level procedure to compute $Time(u, v)$ and $Prob(u, v)$ for all $u, v \in \mathcal{V}$ and then ...

Algorithm 1 Modified Dijkstra's algorithm

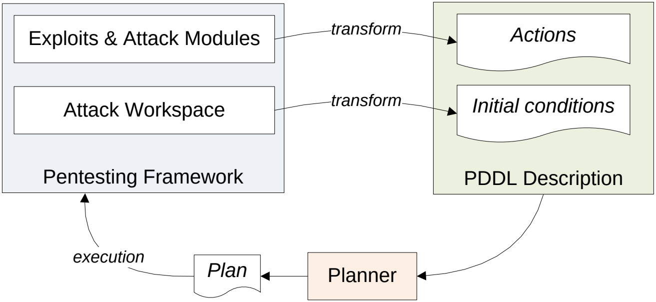
$T[s] = 0,\ P[s] = 1$
$T[v] = +\infty,\ P[v] = 0 \quad \forall v \in \mathcal{V}, v \neq s$
$S \leftarrow \emptyset$
$Q \leftarrow \mathcal{V}$ (where Q is a priority queue)
while $Q \neq \emptyset$ **do**
    $u \leftarrow \arg\min_{x \in Q} T[x]/P[x]$
    $Q \leftarrow Q \backslash \{u\},\ S \leftarrow S \cup \{u\}$
    for all $v \in \mathcal{V} \backslash S$ adjacent to u **do**
        $T' = T[u] + P[u] \times Time(u, v)$
        $P' = P[u] \times Prob(u, v)$
        if $T'/P' < T[v]/P[v]$ **then**
            $T[v] \leftarrow T'$
            $P[v] \leftarrow P'$
return $\langle T, P \rangle$

Agenda

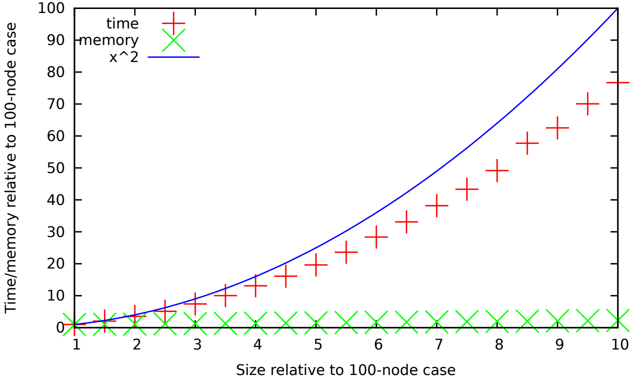
Anatomy of a planning-based attack

Attack Planning, as used in Core Insight Enterprise

[LSR10]; a.k.a. “Cyber Security Domain” [BGHH05]



Experimental results



- Scales up to 1000 machines.
- Planner running time is cuadratic
- Memory consumption is linear.

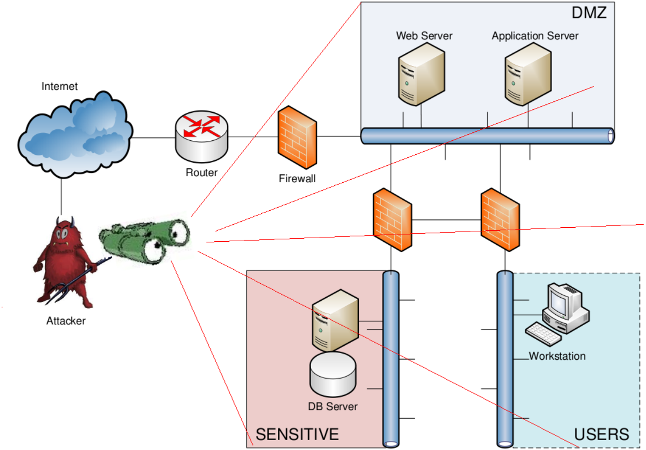
References (for this section)

- [SRL11] An Algorithm to find Optimal Attack Paths in Nondeterministic Scenarios
  - C. Sarraute, G. Richarte, J. Lucangeli
  - *AISec workshop, ACM CCS, Chicago. October 21, 2011.*

Agenda

Anatomy of a real-world attack w/o binoculars

How can this be improved? **Reason about Uncertainty!**



Penetration Testing with Uncertainty

What kind of uncertainty?

Penetration tester has insider knowledge. But can't know *everything!* OS versions, applications installed, . . .

Penetration Testing with Uncertainty

What kind of uncertainty?

Penetration tester has insider knowledge. But can't know *everything!* OS versions, applications installed, . . .

- **Classical solution:**
  (I) gather information (run scans); (II) attack (run exploits)
  - Still simplified: scans don't yield perfect knowledge
  - Exhaustive scans expensive (runtime, traffic)

Penetration Testing with Uncertainty

What kind of uncertainty?

Penetration tester has insider knowledge. But can't know *everything!* OS versions, applications installed, . . .

- **Classical solution:**
  (I) gather information (run scans); (II) attack (run exploits)
  - Still simplified: scans don't yield perfect knowledge
  - Exhaustive scans expensive (runtime, traffic)
- **Our solution:** explicit model of uncertainty in POMDP
  - POMDP plans intelligently mix (I) and (II)
  - Grounds attack planning with uncertainty in formal framework
  - Only related work: neither of these [SRL11]

Penetration Testing with Uncertainty

What kind of uncertainty?

Penetration tester has insider knowledge. But can't know *everything!* OS versions, applications installed, ...

- **Classical solution:**
  (I) gather information (run scans); (II) attack (run exploits)
  - Still simplified: scans don't yield perfect knowledge
  - Exhaustive scans expensive (runtime, traffic)
- **Our solution:** explicit model of uncertainty in POMDP
  - POMDP plans intelligently mix (I) and (II)
  - Grounds attack planning with uncertainty in formal framework
  - Only related work: neither of these [SRL11]
  - Difficulty: make it scale!

Agenda

Markov Decision Process (MDP)

Definition

An *MDP* is a tuple $\langle \mathcal{S}, \mathcal{A}, T, r \rangle$ where:

- $\mathcal{S}$ is the state space
- $\mathcal{A}$ is the action space
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the transition function
  - $T(s, a, s')$ is the probability of coming to state s' when executing action a in state s
- $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function

Definition

Solution: policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$
Objective: maximize expected reward $E\left[\sum_{t=0}^{\infty} r_t \middle| \pi\right]$

Partially Observable MDP (POMDP)

Definition

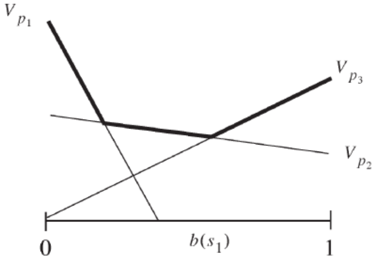
A POMDP is a tuple $\langle \mathcal{S}, \mathcal{A}, T, r, \mathcal{O}, O, b_0 \rangle$ where:

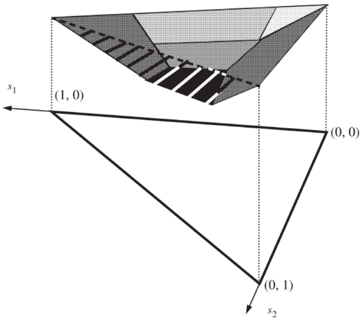
- $\langle \mathcal{S}, \mathcal{A}, T, r \rangle$ is a Markov decision process
- $\mathcal{O}$ is the space of observations
- $O : \mathcal{S} \times \mathcal{A} \times \mathcal{O} \to [0, 1]$ is the observation function
  - $O(s, a, o)$ is the probability of making observation o when executing action a in state s
- b_0 is the initial belief (probability distribution over $\mathcal{S}$)

POMDP Policies

Definition

Solution: policy $\pi : \mathcal{H} \rightarrow \mathcal{A}$ ($\mathcal{H}$: action/observation histories)
Objective: maximize expected reward $E\left[\sum_{t=0}^{\infty} r_t \middle| b_0, \pi\right]$





Equivalent: policy $\pi : \mathcal{B} \rightarrow \mathcal{A}$ where $\mathcal{B} = \Pi(\mathcal{S})$

Solving POMDPs

- **Is it hard?**
  - $\mathcal{S}$: all states (= all possible configurations)
  - Belief states *b*: probability distributions over $\mathcal{S}$
  - ...and we need to *reason* about this stuff!
- **How to do it?**
  - Here: SARSOP [KHL08]
  - Approximate belief value based on selected belief states (get hyperplane for each, compute upper envelope)
- **What about scaling?**
  - Using out-of-the-box planners: **Bad!**
  - Proposal: use in “1-machine case”, design global solution by decomposition + approximation

Agenda

Birds-Eye View

- **States**
  - Network structure static and fully known
  - Combinations of configuration parameters . . .
  - . . . as relevant to modeled exploits!
- **Actions**
  - Exploits: succeed/fail depending on state
  - Scans: return observation depending on state
  - Both are deterministic!
- **Rewards**
  - $r = V - T - D$: value of computer, runtime, detection risk
  - V: human decision; T, D: estimate using statistics
- **Initial belief**
  - Probability distribution over configurations
    $\Longrightarrow$ uncertainty from point of view of pentesting tool

Example: Actions

```
actions :

Probe-M0-p445
OSDetect-M0

Exploit-M0-win2000-SMB
Exploit-M0-win2003-SMB
Exploit-M0-winXPsp2-SMB

Terminate
```

"Terminate" action: give planner the choice to "give up" if expected costs outweigh expected reward

Example: States (1 Machine)

```
states :

M0-win2000
M0-win2000-p445
M0-win2000-p445-SMB
M0-win2000-p445-SMB-vuln
M0-win2000-p445-SMB-agent

M0-win2003
M0-win2003-p445
M0-win2003-p445-SMB
M0-win2003-p445-SMB-vuln
M0-win2003-p445-SMB-agent

M0-winXPsp2
M0-winXPsp2-p445
M0-winXPsp2-p445-SMB
M0-winXPsp2-p445-SMB-vuln
M0-winXPsp2-p445-SMB-agent

M0-winXPsp3
M0-winXPsp3-p445
M0-winXPsp3-p445-SMB

terminal
```

Example: Scans – OS Detection

```
O: OSDetect-M0: M0-win2000                 : win 1
O: OSDetect-M0: M0-win2000-p445            : win 1
...
O: OSDetect-M0: M0-win2003                 : win 1
O: OSDetect-M0: M0-win2003-p445            : win 1
...

O: OSDetect-M0: M0-winXPsp2                 : winxp 1
O: OSDetect-M0: M0-winXPsp2-p445            : winxp 1
...
O: OSDetect-M0: M0-winXPsp3                 : winxp 1
O: OSDetect-M0: M0-winXPsp3-p445            : winxp 1
...
```

Example: Exploit SAMBA Server on Port 445

```
T: Exploit-M0-win2003-SMB identity
T: Exploit-M0-win2003-SMB: M0-win2003-p445-SMB-vuln
                               : * 0
T: Exploit-M0-win2003-SMB: M0-win2003-p445-SMB-vuln
                               : M0-win2003-p445-SMB-agent 1

O: Exploit-M0-win2003-SMB: * : * 0
O: Exploit-M0-win2003-SMB: * : no-agent 1
O: Exploit-M0-win2003-SMB: M0-win2003-p445-SMB-agent
                               : agent-installed 1
```

Agenda

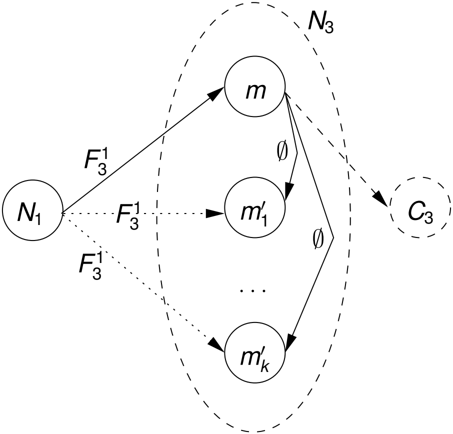
Decomposition – Level 4

Attack machine M_2 from machine M_1:

- We use out-of-the-box POMDP planners.
- In our experiments: we use SARSOP.

Decomposition – Level 3

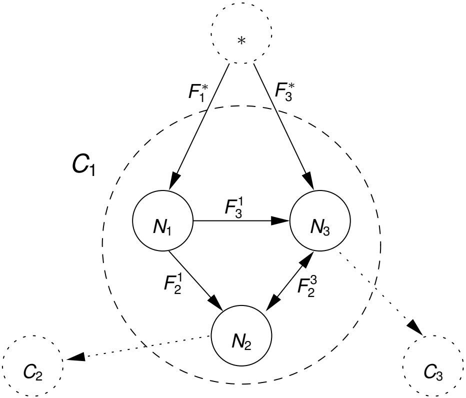
Group machines into Logical Subnetworks N.



Attacking N_3 from N_1, using m first.

Decomposition – Level 2

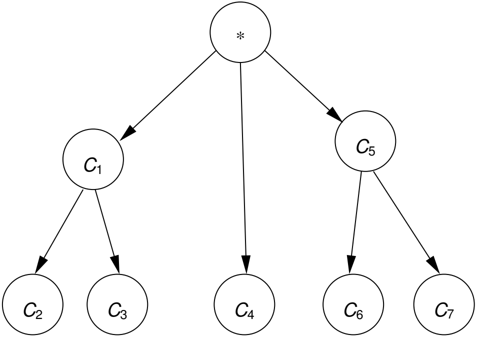
Group the subnetworks into Biconnected Components C.



Paths for attacking C_1.

Decomposition – Level 1

What you get in the end: a beautiful and simple tree.



LN as tree of components *C*.

Agenda

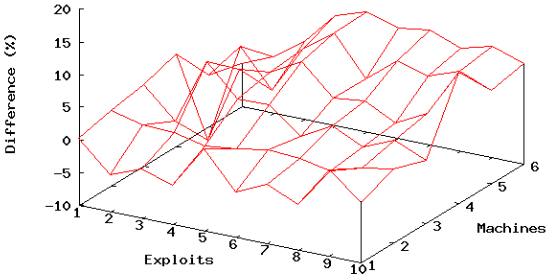
Test Examples

Problem generator with 3 parameters:

- Number M of machines in network
  Agent on machine M_0, M “behind” M_0 in fully connected network
- Number E of exploits considered
  $E \geq M$, distributed evenly across machines
- Time delay T (days) since last pentest
  Update parameters estimated by hand

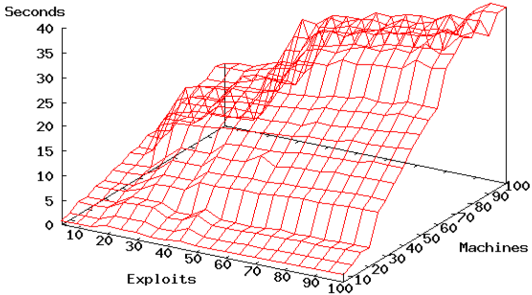
Here: $1 \leq M \leq 100$; $1 \leq E \leq 100$; $0 \leq T \leq 200$

Results I



Attack quality comparison: Empirical results for the 4AL decomposition compared to a global POMDP model.

Results II



Running time of the 4AL decomposition algorithm.

References (for this section)

Joint work with researchers at INRIA (Nancy, France)

Jörg Hoffmann, author of FF [Hof01] and Metric-FF [Hof02], reference tools for “classical” planning.

Olivier Buffet, author of books and tools on Markov decision process [SB10].

- [SBH11] Penetration Testing == POMDP Solving?
  - *SecArt’11 (Workshop on Intelligent Security), IJCAI’11 Conference, Barcelona. July 16-22, 2011.*
- And a new paper to be published in AAAI 2012 (Toronto, 22 - 26 July 2012)

Agenda

1. Motivation
2. On Exploit Quality Metrics
3. The Search for an Efficient Solution
   - Planning for dummies
   - Two primitives
   - Using the primitives in a Network Graph
   - Integration with a Pentesting Tool
4. The Search for a Better Model
   - POMDPs
   - Penetration Testing as POMDPs
   - Decomposition in 4 Abstraction Levels
   - Experiments
5. Conclusion

Probabilistic Planner Summary

First direction . . . we have presented:

- An **attack model** based on exploits metrics:
  - Average running time
  - Probability of success
  - Details of the vulnerable platform (OS and application versions)
  - Connectivity requirements.
- An efficient planning solution, **integrated** to a penetration testing framework.
- An **evaluation of our implementation** that shows the feasability of planning and verifying attacks in **real-life scenarios**.

POMDP Model Summary

Second direction . . . reasoning under uncertainty

(a) Beliefs: likelihood of particular vulnerabilities
$\Longrightarrow$ order exploits by promise

(b) Belief transitions: update "promise" as more information comes in
$\Longrightarrow$ order exploits dynamically

(c) Belief transitions vs. rewards (time/risk): trade-off observation gain against its cost
$\Longrightarrow$ apply scans only where needed/profitable

POMDP Model: What have we gained?

- More accurate model of attack planning with uncertainty
- Can deliver better plans thus more effective pentesting
  - Policy = stronger notion of plan
  - Contemplates all possible histories of actions / observations.
- The 4AL decomposition provides a reasonable scaling.

Bridging the language gap

- Separate the problem from potential solutions.
- Communicate our problem to the AI / Planning community
  $\longrightarrow$ they're looking for practical applications!

- Solving: PoC implementation shows feasibility
  Scaling to large networks $\implies$ decompose/approximate with 1-target-machine cases

- Basic AI: these POMDPs have particular properties ...
  $\longrightarrow$ open path for further research

That's all folks!

Thanks for your attention!
Questions?

carlos @ coresecurity . com
http://corelabs.coresecurity.com/

References I

Ivan Arce.
On the quality of exploit code: An evaluation of publicly available exploit code.
In *RSA Security Conference*, San Francisco, CA, 2005.

Mark S. Boddy, Johnathan Gohde, Thomas Haigh, and Steven A. Harp.
Course of action generation for cyber security using classical planning.
In *Proc. of ICAPS'05*, 2005.

Jörg Hoffmann.
FF: The fast-forward planning system.
AI magazine, 22(3):57, 2001.

Jörg Hoffmann.
Extending FF to numerical state variables.
In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI-02)*, pages 571–575, 2002.

References II

H. Kurniawati, D. Hsu, and W. Lee.
SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces.
In *RSS IV*, 2008.

Jorge Lucangeli, Carlos Sarraute, and Gerardo Richarte.
Attack Planning in the Real World.
In *Workshop on Intelligent Security (SecArt 2010)*, 2010.

Carlos Sarraute.
New algorithms for attack planning.
In *FRHACK Conference, Besançon, France*, 2009.

Carlos Sarraute.
Probabilistic Attack Planning in Network + WebApps Scenarios.
In *H2HC Conference, Sao Paulo, Brazil*, 2009.

References III

O. Sigaud and O. Buffet, editors.
Markov Decision Processes and Artificial Intelligence.
ISTE - Wiley, 2010.

Carlos Sarraute, Olivier Buffet, and Jörg Hoffmann.
Penetration testing == POMDP planning?
In *Proceedings of the 3rd Workshop on Intelligent Security (SecArt'11), at IJCAI*, 2011.

Carlos Sarraute, Gerardo Richarte, and Jorge Lucangeli.
An algorithm to find optimal attack paths in nondeterministic scenarios.
In *Proceedings of the ACM Workshop on Artificial Intelligence and Security (AISec'11)*, 2011.