Probabilistic Attack Planning in Network + WebApps Scenarios

Carlos Sarraute

Core Security Technologies

and *Ph.D. program in Informatics Engineering, ITBA*

H2HC – Nov 28/29, 2009

Brief presentation

- My company: Core Security Technologies
  - Boston (USA)
    - marketing and sales
  - Buenos Aires (Argentina)
    - research and development
- I have worked in Corelabs since 2000
  - that's the research lab (in Buenos Aires)
  - coordinate research activities (e.g. Bugweek) and publication of advisories
  - focus area: applying Artificial Intelligence techniques to solve problems from the security field

Outline

- Introduction
- The Attack Model
- Our family of Agents
- Fast probabilistic algorithms

Introduction

Why do we need automation?

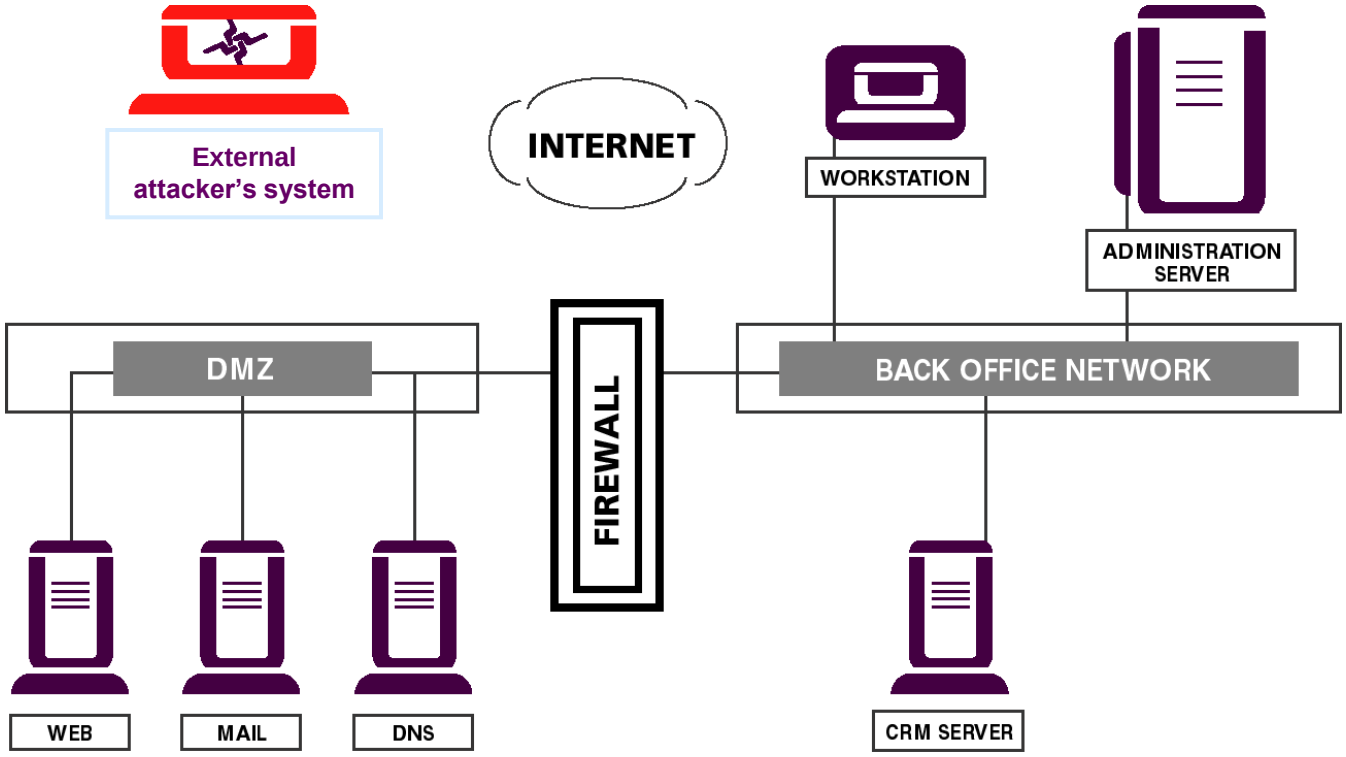
- Evolution of pentesting
  - Attacks are evolving
  - Organizations are evolving
    - technological complexity
    - infrastructure complexity
  - Manual pentesting requires more expertise and time
  - Continuous pentesting

- Pentesting tools are evolving
  - Metasploit (open source)
  - Immunity Canvas and Core Impact (commercial)

Increase pentesting scale

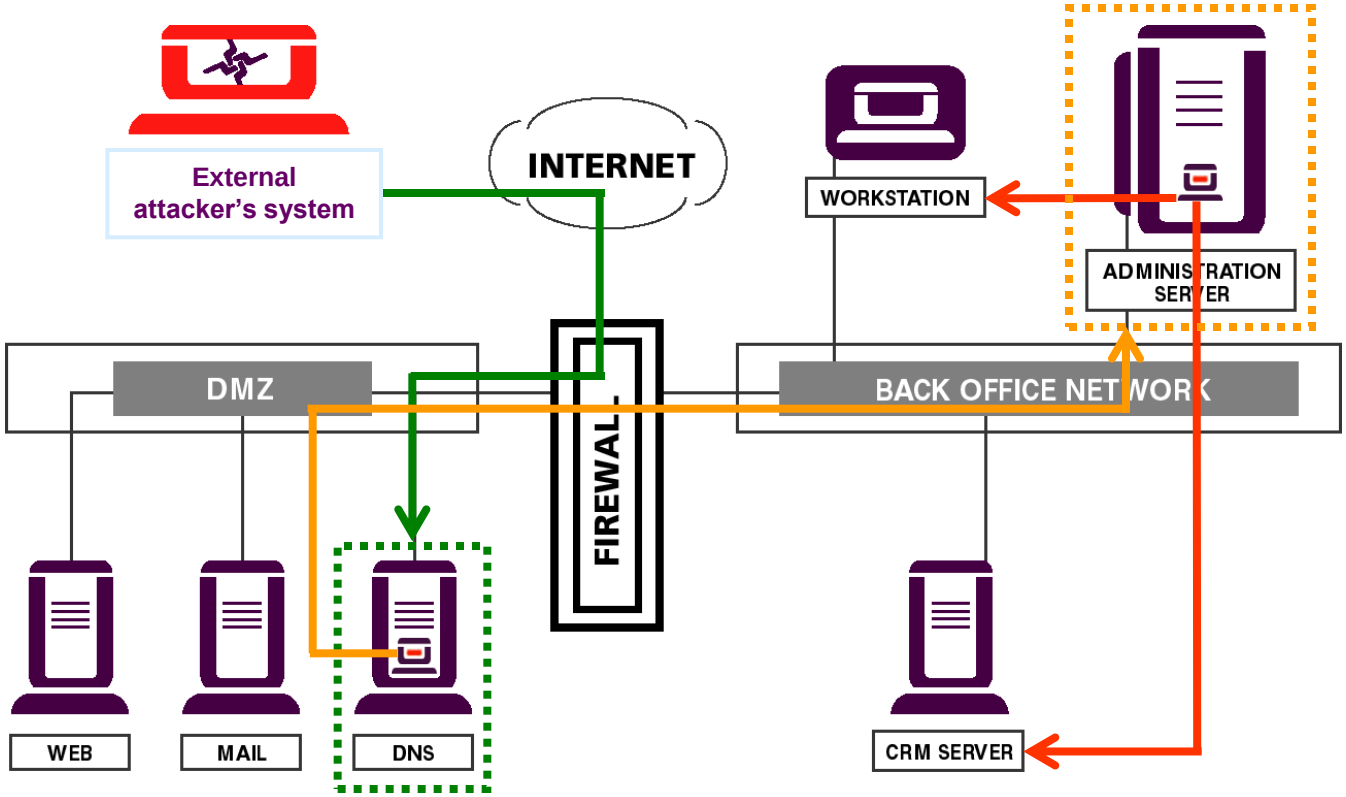
- Example: pentest a network with 500 machines
  - limited human resources
  - bounded time frame
  - pentest mimics attacks which doesn't have those restrictions
- Automating repetitive tasks liberates time for
  - research / creative work
  - training / be up-to-date
  - produce more complex attacks
- Make it more accessible
  - The admin can test his own network

Sample pentest scenario



Anatomy of a real-world attack

A sophisticated real-world attacker will leverage trust relationships to gain access to more valuable information assets



Base camp

A target server is attacked and compromised

The acquired server is used as vantage point to penetrate the corporate net

Further attacks are performed as an internal user

The Attack Model

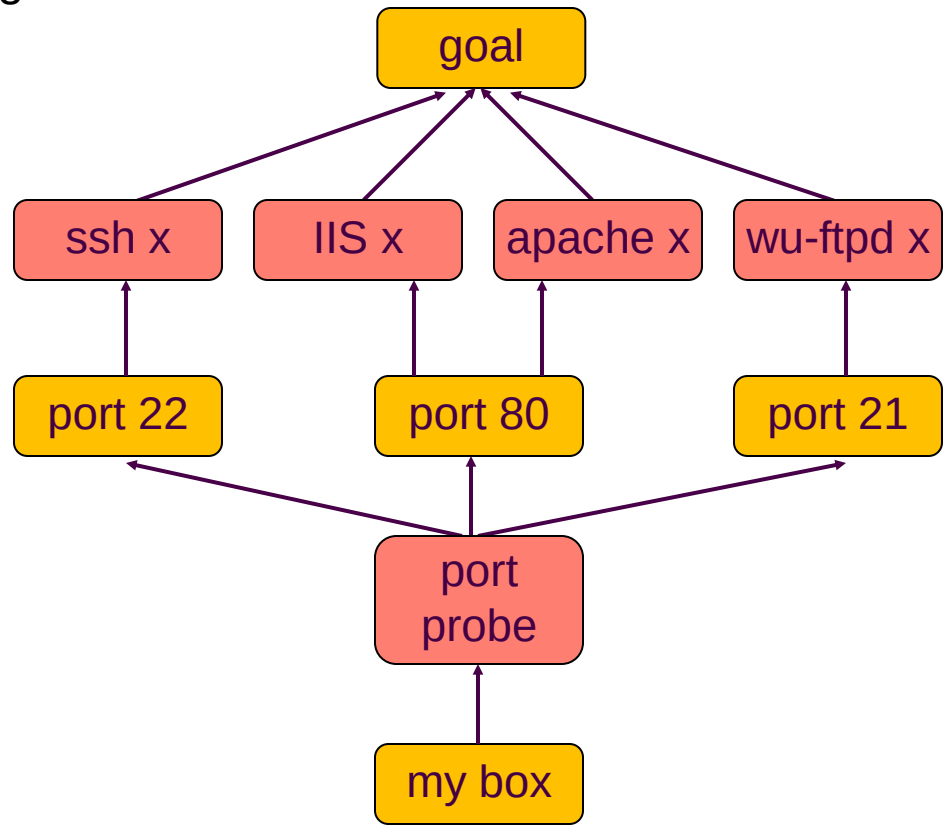
Example of attack planning

Goal: To gain control of any host in target network

Assets: Target's IP address
Control of my box
A set of IG tools and exploits

Actions:
test if a given port is open (port probe)
exploit ssh (on an OpenBSD)
exploit wu-ftpd (on a Linux)
exploit IIS (on a Windows)
exploit apache (on a Linux)

Plan:
Probe only ports 22, 80 and 21.
Probe port 80 first!
As soon as a port is found open, run an exploit.
Keep probing other ports only if exploit fails.



The model components

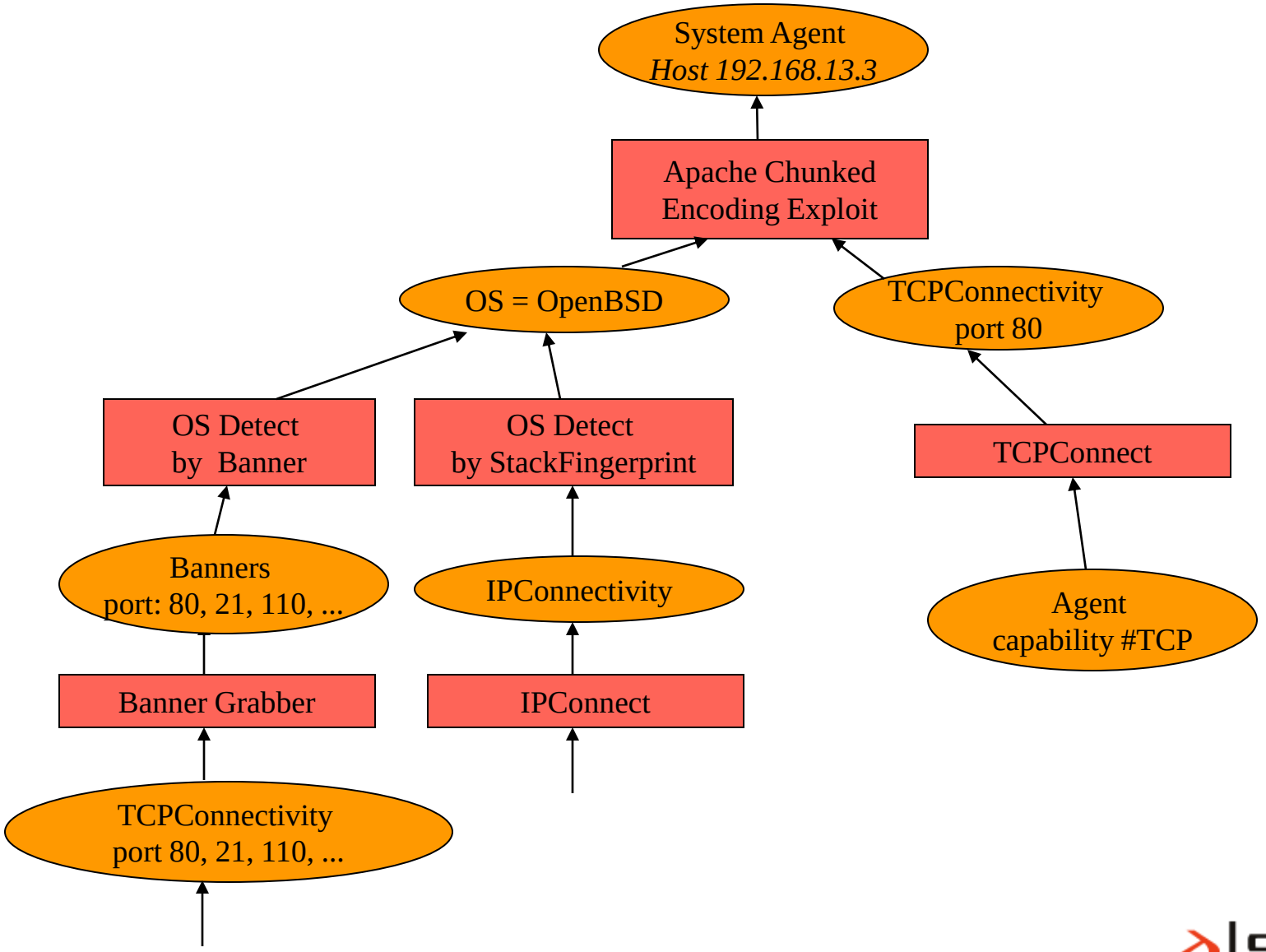
- Goals
  - Objectives of the attack
    - Obtain credit card numbers from the Database server
- Assets
  - Anything an attacker may need during the attack
    - OperatingSystemAsset, TCPConnectivityAsset and AgentAsset
- Actions
  - Atomic step that can be part of an attack
    - An exploit, a TCP connection and an OS identification method
- Agents: perform the actions

Attack Graph nodes

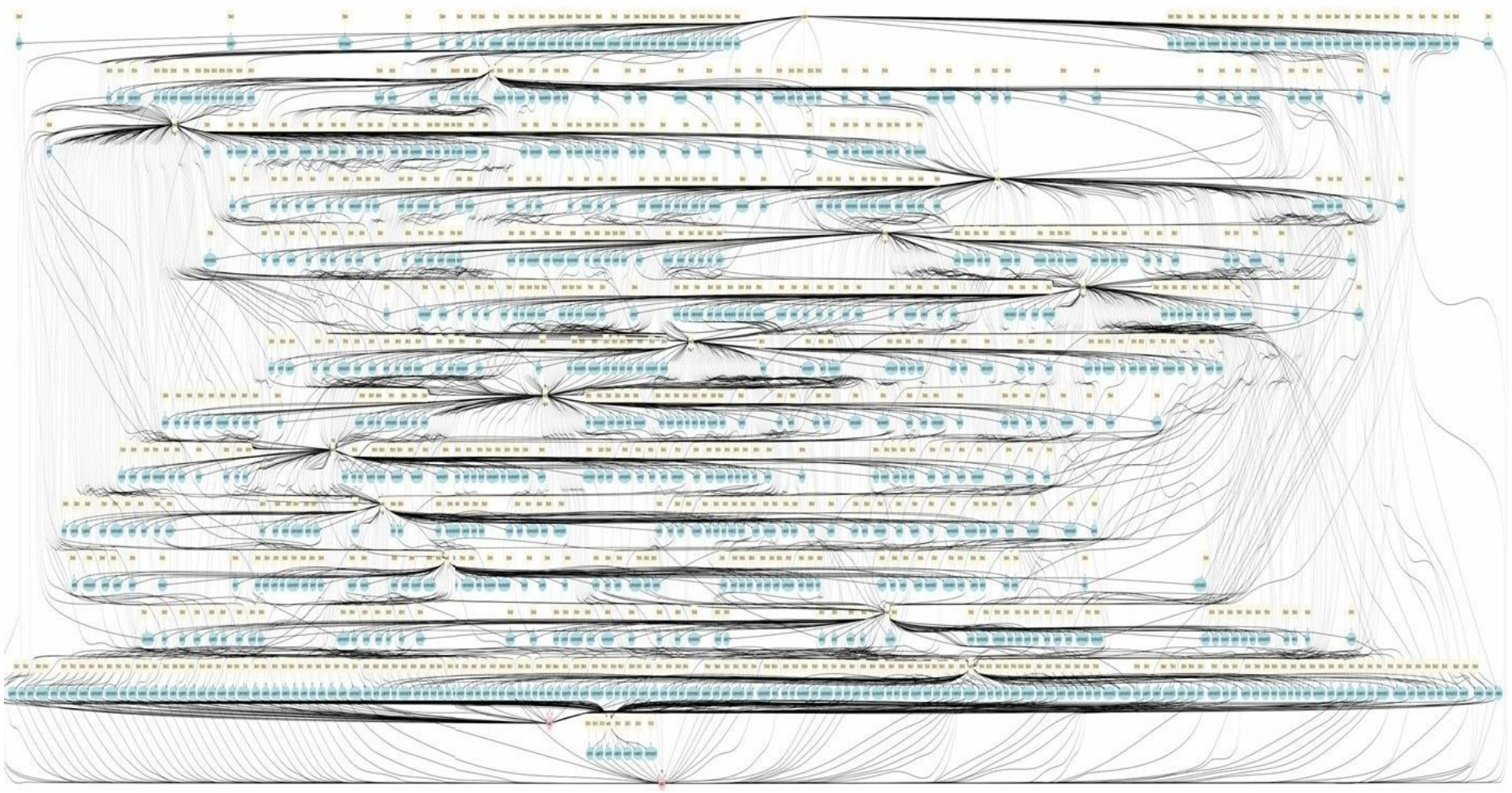
- The graph nodes are Actions and Assets

- Every action has an associated result
  - an Exploit gives as result an Agent on the target machine
- Actions have requirements (preconditions or subgoals)
  - Exploits are platform dependent and require knowledge of the Operating System of the target before execution
  - an HTTP Exploit requires an open port (and connectivity)

Alternated layers of actions and assets



An Attack Graph, a bit more real



From Noel – Jajodia: "Managing Attack Graph Complexity Through Visual Hierarchical Aggregation"

Cost of actions

- Add realism and increase difficulty of planning problem
- Actions have an associated cost function
  - actions produce **noise**
    - network traffic
    - log lines
    - IDS events
  - expected **running time**
  - *planning: requires numerical effects*
- Actions have a **probability of success**
  - *requires probabilistic planning*

Our family of Agents

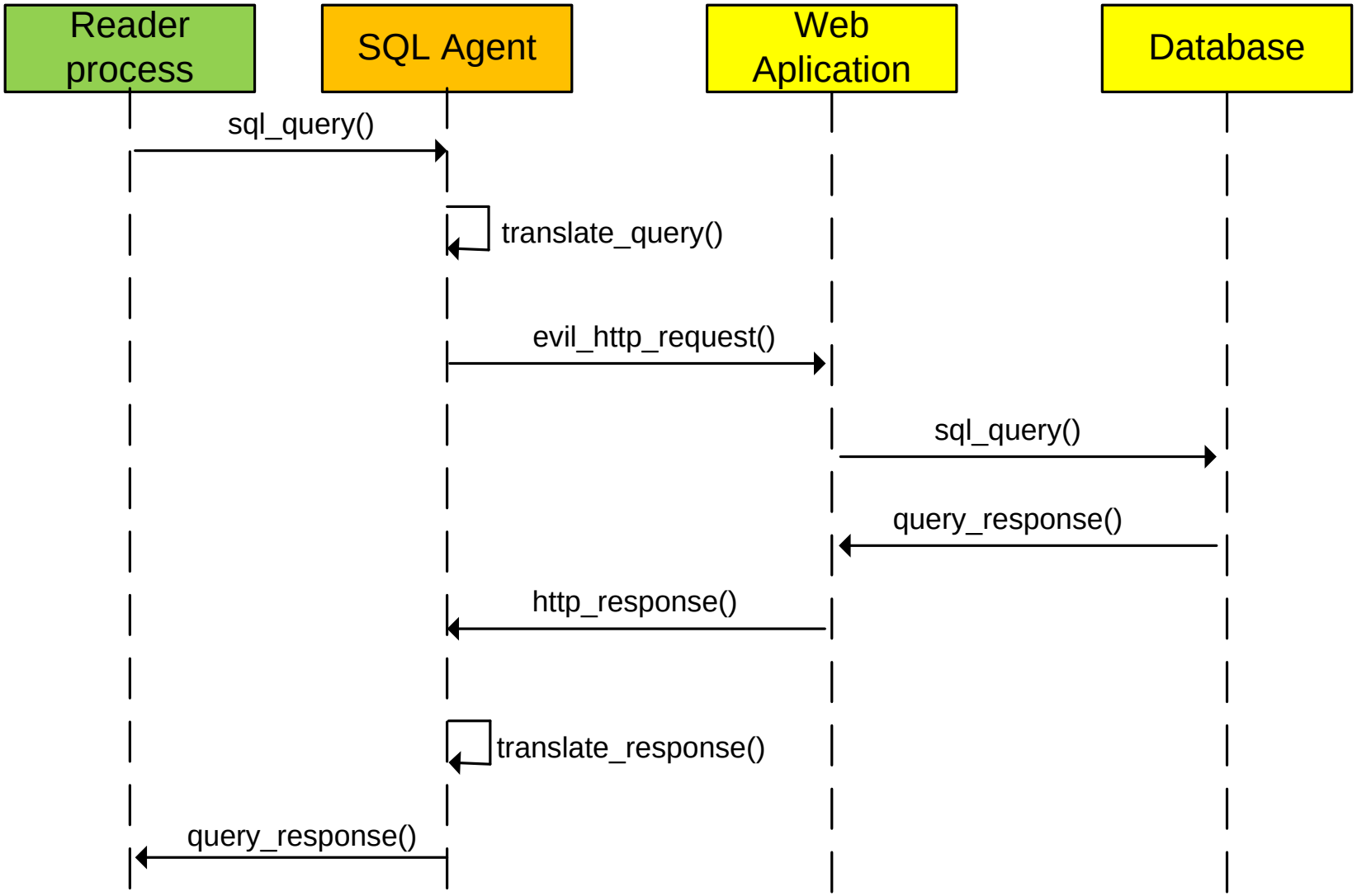
System Agent

- Exploiting a binary vulnerability gives a System Agent
- In the proxy-call architecture
  - the agent is a small proxy-call server
  - executes system calls locally
  - and sends the result of the execution
- Capabilities:
  - access to the target filesystem
  - access to the network
  - transparent pivoting (allows chaining of agents)
- Ref: Rodrigo Branco and Felipe Balestra's presentation in H2HC 2006: "Syscall Proxying || Pivoting Systems"

SQL Agent

- Exploits a SQL Injection on a web application
  - *http://vulnerable.com/vuln.php?field='SELECT+customerId,customerName+FROM+customers--*
- Is able to submit SQL queries to the remote database, and receive the answer
  - much like the System Agent does with system calls.
- We can think that we are "installing" the SQL agent on the remote machine
  - in fact it means that we have found an exploit that the agent can use to translate SQL commands to a given target

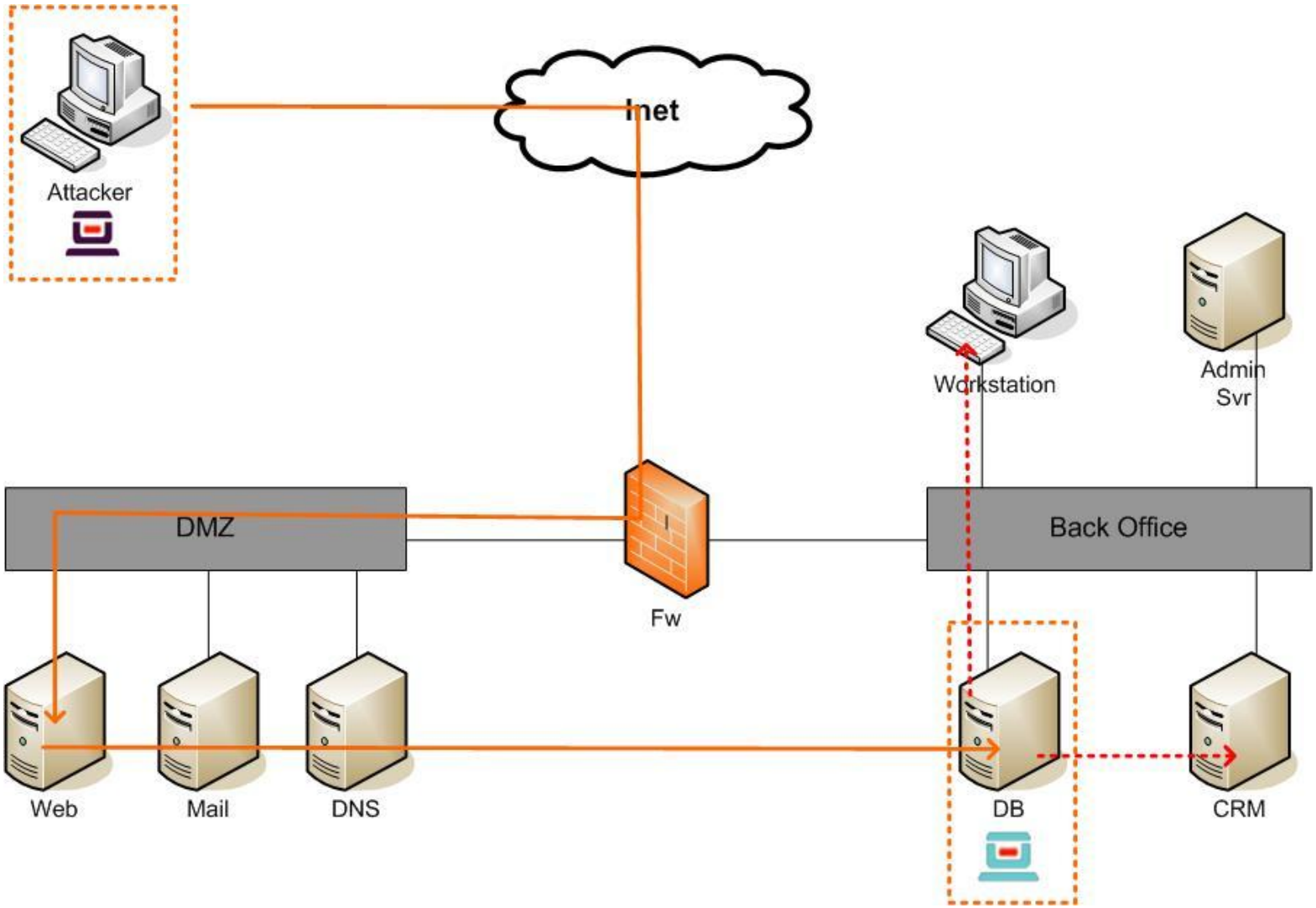
SQL Agent in action



SQL Agent benefits

- Abstract the gained capability from the complexity of the vulnerability
  - exploitable query length,
  - filtered characters,
  - column type,
  - bandwidth, etc.
- Presents the attacker with a homogenous programming interface
  - independent from the vulnerabilitie's restrictions y the DB backend
- Fits nicely into the Attack Planning Model

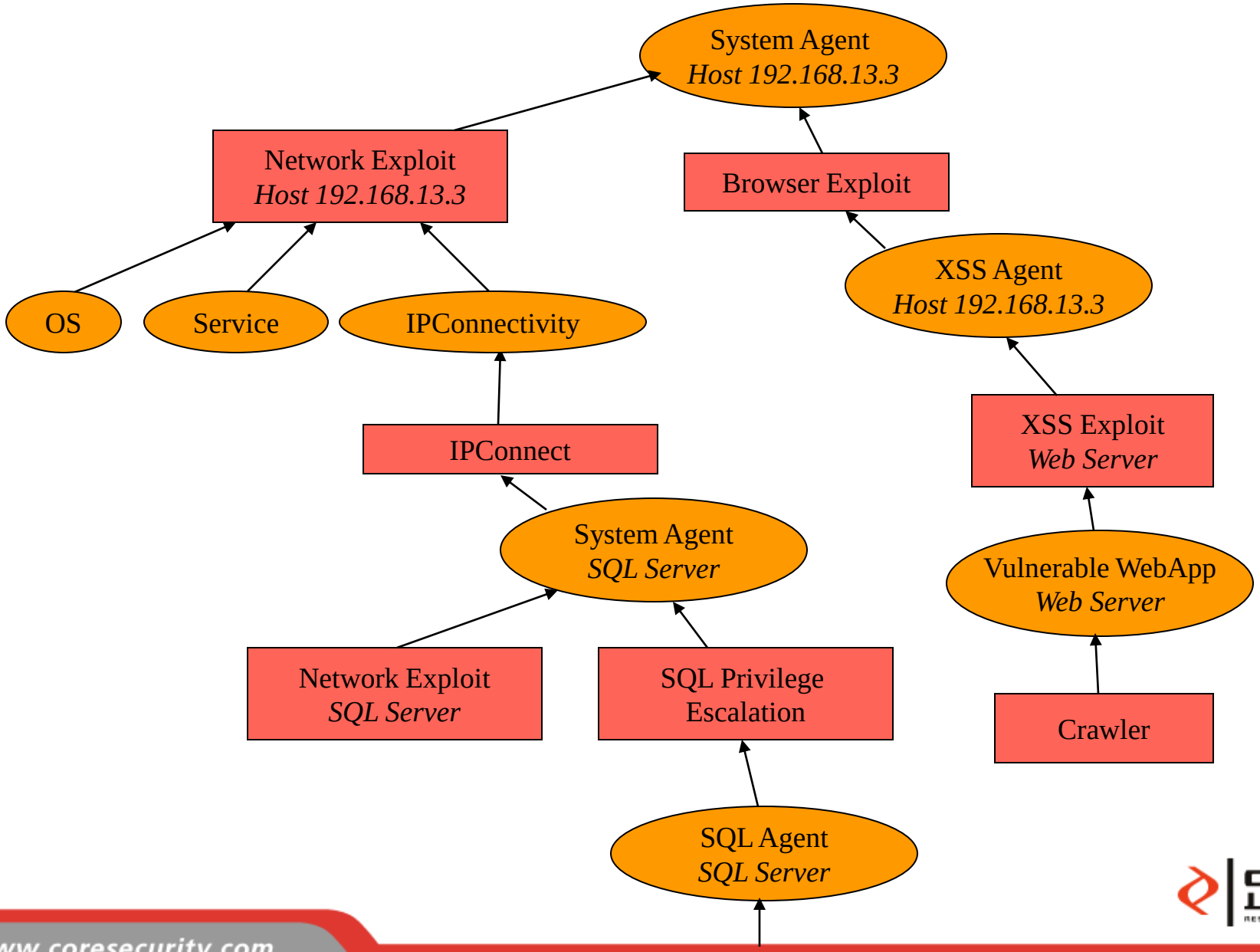
Sample scenario involving SQL Agent



XSS Agent

- Exploits a Cross Site Scripting vulnerability to inject JavaScript code
  - <script src=http://mysite/egg.js></script>
- The agent handles the attacker's web server
- Provides the simplified API to send actions to the owned browser.
- Sample actions:
  - Port scanners
  - Steal credentials (cookies)
  - JavaScript console

Attack Graph involving XSS and SQL Agents



Fast algorithm for Probabilistic Planning

Scenario 1: one goal, many exploits

- Attacker wants to gain access to the credit cards stored in database server *H*
- Attacker has a set of *n* remote exploits that he can launch against that server.

- The exploits result in the installation of a system agent when successful. The attacker
  - estimates probability of success based on the information already gathered
  - knows expected running time of each exploit

How many exploits?

- Automation module of Core Impact
  - has 6 years of evolution

Modules by category

Category	Modules
Remote Exploits	177
Local Exploits	61
Client-side Exploits	140
Denial-of-Service (DoS) Exploits	27
Utilities	158
Total	**563**

Target entry points

Operating System	Exploits	Unique Targets
Windows Vista	42	116
Windows 2003	113	743
Windows XP	216	1236
Windows 2000	229	2403
Windows NT	19	84
Linux	155	478
Solaris	32	90
AIX	3	5
Mac OS X	9	53
OpenBSD	15	41
FreeBSD	7	17
Total	**840**	**5266**

- Deals with 840 exploits, targeting 5266 unique targets
- Tested on Class B networks with 512 hosts

How to measure time and probability?

- Measure results of exploit executions in testing lab
  - 748 virtual machines in Core's testing lab
  - different OS and installed applications

- Get feedback from the users
  - anonymized feedback program in Impact
    - sensitive data is filtered out before sending it
  - natural option for Metasploit (in my opinion)

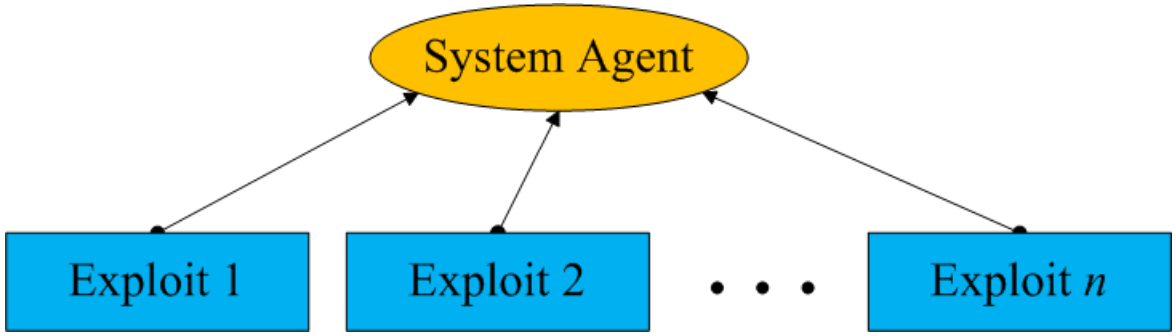
Problem 1: one goal, many actions

- Let T be a fixed goal
- Let $A_1, \ldots, A_n$ be a set of n independent actions whose result is T.
  - each action A_k has a probability of success p_k
  - and expected running time t_k
- Actions are executed until an action provides the goal T.

Task: Find the order of execution to minimize the expected total running time.

	time	probability
action1	20 s	0.55
action2	30 s	0.85
action3	3 s	0.02
action4	120 s	0.95

Expected values



- If the actions are executed in the order $A_1, \ldots, A_n$
- The expected running time is:

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

- The probability of success is:

$$P_{\{1\ldots n\}} = p_1 + \overline{p_1}\, p_2 + \overline{p_1}\,\overline{p_2}\, p_3 + \ldots + \overline{p_1} \ldots \overline{p_{n-1}}\, p_n$$

A nice Lemma

- Let A_1, …, A_n be actions such that

$$\frac{t_1}{p_1} < \frac{t_2}{p_2} < \ldots < \frac{t_n}{p_n}$$

- Then

$$\frac{T_{\{1\ldots n-1\}}}{P_{\{1\ldots n-1\}}} < \frac{t_n}{p_n}$$

- Proof: by induction.

Proposition 1 (solution)

- A solution to Problem 1 is to order the actions according to the coefficient t_k / p_k, and execute them in that order.

- The computational complexity of this solution is

$$O (n \log n)$$

- In our small example:

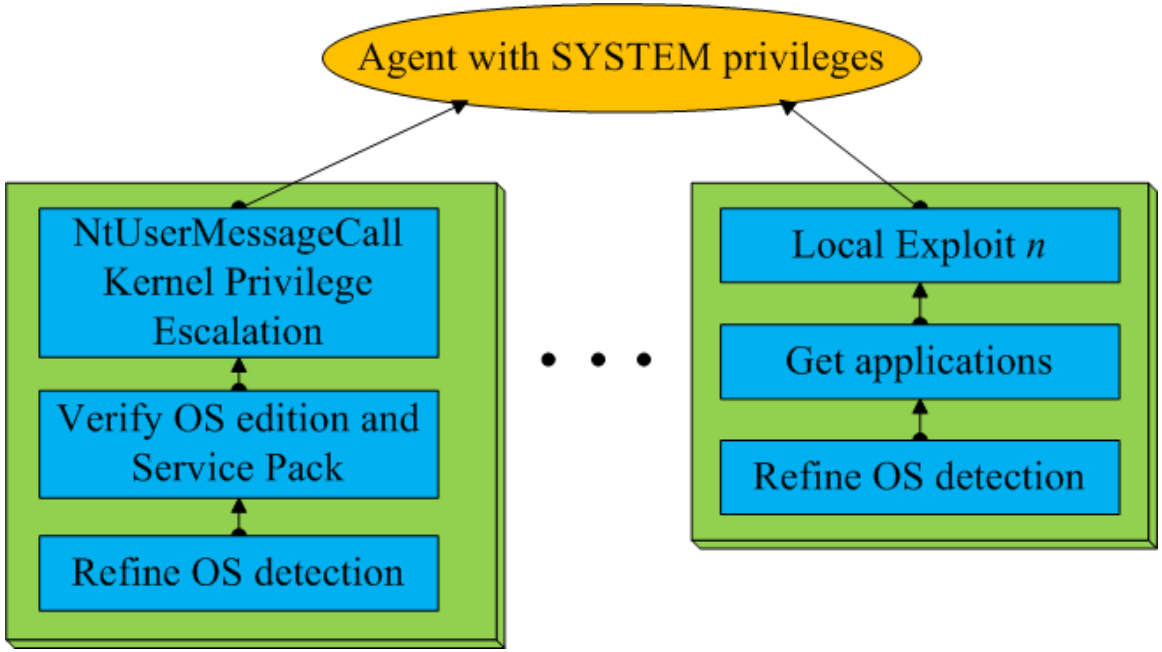
	time	probability	coefficient	order
action1	20 s	0.55	36.36	2
action2	30 s	0.85	35.29	1
action3	3 s	0.02	150.00	4
action4	120 s	0.95	126.31	3

Problem 2: multiple strategies

- **strategy:** group of actions that must be executed in a specific order.
- The strategies are a way to incorporate the expert knowledge of the attacker in the planning system
- Cf. the opening moves in chess or Go

Strategy example

- Example: the attacker has an agent with low privileges on host *H* and his goal is to obtain system privileges



Problem 2: one goal, many strategies

- Let T be a fixed goal, and let $G_1, \ldots, G_n$ be a set of n strategies.
- Each strategy G_k is composed by a **group of ordered actions**.
- If all the actions in a group are successful, the strategy fulfills the goal T.

- **Task:** Minimize the expected total running time.

Proposition 2 (solution)

- Calculate expected running time of each group

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

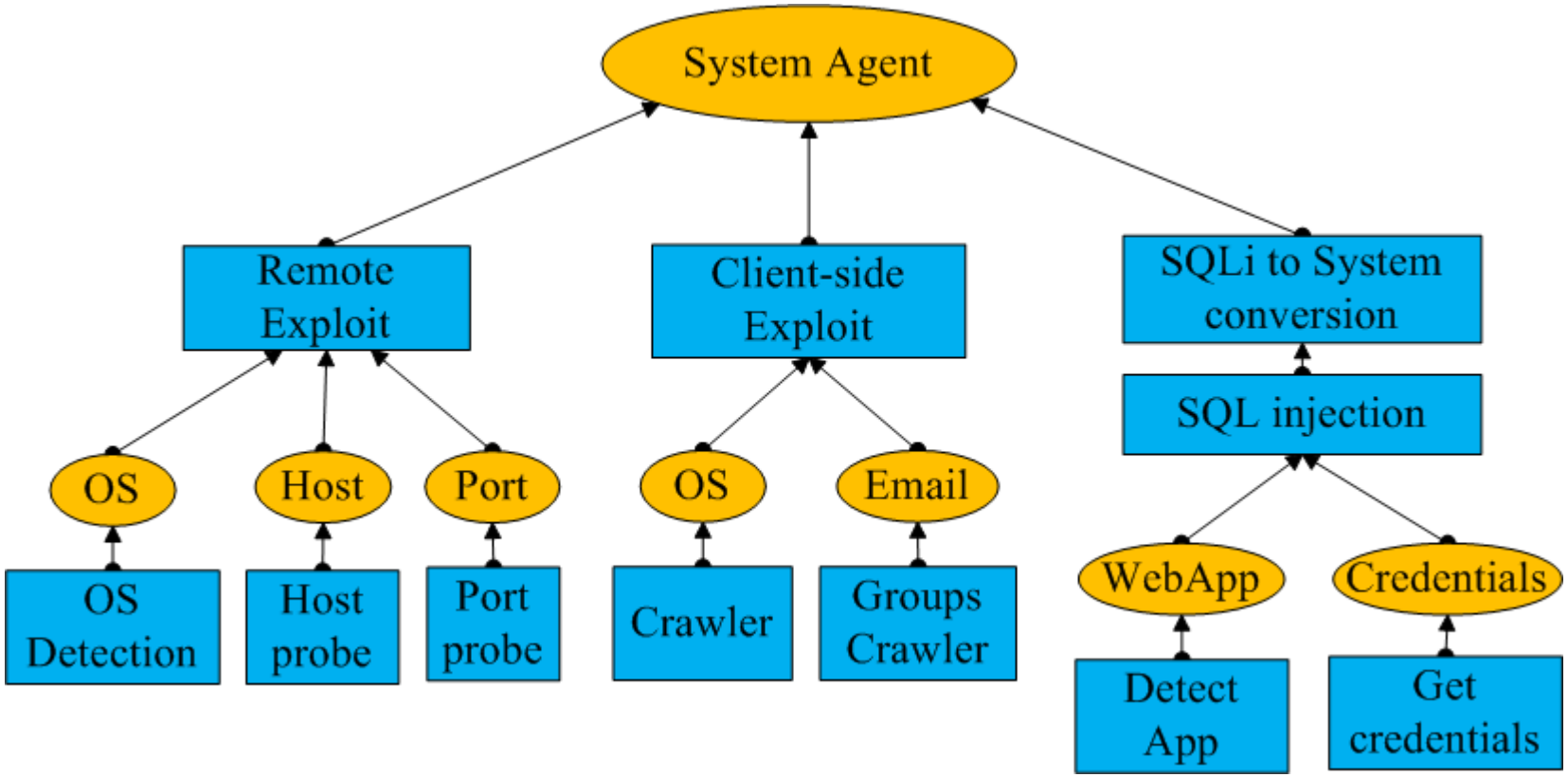
- Calculate probability of success

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

- Sort the strategies according to T_G / P_G
- In each group execute the actions until an action fails
  - this is the technical part of the demonstration

Problem 3: two layers attack tree

- Groups of actions bounded by an **AND relation**
  - the order of actions is **not** specified
  - in previous problem the order was fixed



Proposition 3 (solution)

- How to order the actions in each group?

- **Lemma:** To minimize the expected total running time, the actions must be ordered according to the coefficient

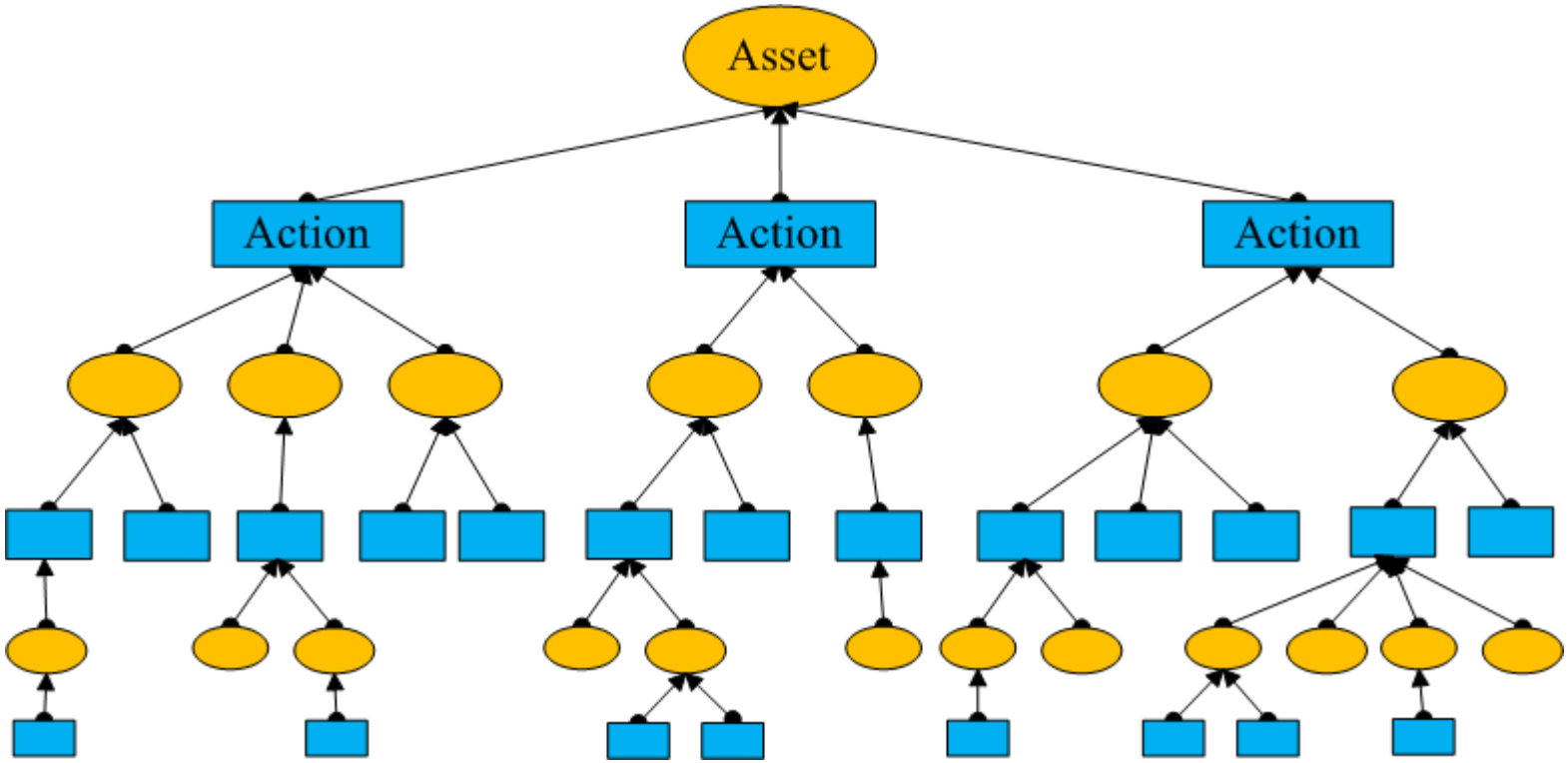
$$t_k / (1 - p_k)$$

- **Intuition:** the actions that have higher probability of failure have higher priority, since a failure ends the execution of the group.

Dynamic Replanning

Problem 4: attack tree

- Attack tree, alternating Assets and Actions



Proposition 4 (solution)

- Compose all previous algorithms
- **AND group:** can be considered as a single node with probability P_G and execution time T_G
- **OR group:** the node that minimizes the t/p coefficient will be executed first
  - considered as the cost of the group in a single step plan.
- By iteratively reducing groups of nodes, we build a single path of execution

Dynamic replanning

- After executing a step of the plan, the costs may be modified and the shape of the graph may vary.

- This is where dynamic replanning comes in.
  - Since the planning algorithm is very efficient, we can replan after each execution
  - and build a new path of execution.

Conclusion

Summary

- Attack planning – from the attacker's point of view
  - consider all the steps of an attack, not only exploits
  - model the attacker's knowledge of the world
- Extension to classic Attack Graphs
  - numerical effects
    - expected running time
  - probabilistic effects
    - probability of success
- Fast algorithm for Probabilistic Attack Planning
  - works in a relevant part of real-world scenarios
  - demonstrations that the solution is optimal in specified scenarios

New research direction

- During the last years, the difficulties in our research were related to the exponential nature of planning algorithms
  - especially in the probabilistic setting
- Our efforts were directed toward the aggregation of nodes and simplification of the graphs
  - to tame the size and complexity of the problem
- Having a very efficient algorithm in our toolbox gives us a new direction of research:
  - refine the model
  - break down the actions into smaller parts
  - without fear of producing an unsolvable problem.

Finer analysis of exploits

- A future step: divide the exploits into basic components.
- This decomposition gives a better probability distribution of the exploit execution
- Example: Debian OpenSSL Predictable Random Number Generation Exploit
  - brute forces the 32,767 possible keys.
  - each iteration is considered as a basic action
  - some keys are more probable than others
- Finer level of control over the exploit execution
  - produces gains in the total execution time

Thank you!

Carlos Sarraute → carlos@coresecurity.com

http://corelabs.coresecurity.com