Attack analysis

A system model

- The discovery of attacks against a system requires the definition of a system model that describes
  - System components
  - System interconnection structure
  - Component vulnerabilities
  - Attacks (simple steps)
- The level of description and of detail obviously depends upon the accuracy we aim to achieve

A system model

- The discovery of attacks against a system requires the definition of a system model that describes
  - System components
  - System interconnection structure
  - Component vulnerabilities
  - Attacks (simple steps)
- The level of description and of detail obviously depends upon the accuracy we aim to achieve

Modelling an attack - I

- Any attack can be modelled through (at least) six attributes
-

1. Precondition

•rights on system objects

• resources

• competences and info

2. Post condition

•rights on system objects

•

3. enabling vulns (component, vulnerabilities)

4. actions to be executed

Modelling an attack - II

- The attack post condition is the set of rights the attacker owns if the attack is successful
- The postcondition always include the preconditon (monotone right acquisition)
- The actions to be executed include
  - Human actions
  - Program execution
- Fully automated attack = no human action is required
- Noise = events the attack generates and that enable the detection of the attack

Example -I

- To implement a buffer overflow, one needs
  - To invoke a procedure (rights)
  - To write a parameter that includes the program to be executed (know how)
  - To know the memory map to determine the size of the parameter to overflow the stack (info)
  - Fully automated attack
  - Success probability = depends on controls in the attacked system

Example -II

- If the attack is successful, the injected program is executed as root and it can access any system resource
- The attack noise is a function of the checks that the target system executes and that make it possible to detect the attack
- The checks influence both the success probability and the noise as they can only discover (log) or also prevent (type -canary) the attack

Attack taxonomies

- Several alternative taxonomies that are focused on just one feature/attribute
  - Enabling vuln
  - The agent that can implement the attack
  - The impact produced by the attack
  - The target component
- All these properties are important but a risk assessment may be focused on other properties or on several of these features

An example of an elementary attack taxonomy

1. Buffer/stack/heap overflow
2. Exchanged information is illegally read (sniffing)
3. Some of the legal messages of a legal user are repeated (replay attack)
4. Interface operations are invoked in an unexpected order (interface attack)
5. Interception and manipulation of information exchanged between two entities (man-in-the-middle)
6. Information flows are diverted
7. Time-to-use Time-to-check (Race condition)
8. XSS (cross site scripting)
9. Covert channel
10. Impersonating
    - A user
    - A machine (IP spoofing, DNS spoofing, Cache poisoning)
    - A connection (connection stealing/insertion)

Covert Channel

Attack against Bell – La Padula security policy

Cryptographic attacks

A dedicated taxonomy

a) Brute force attack
b) Differential cryptanalysis
c) Linear cryptanalysis
d) Meet-in-the-middle attack
e) Chosen-ciphertext attack
f) Chosen-plaintext attack
g) Ciphertext-only attack
h)
i)

h) Known-plaintext attack
i) Power analysis
j) Timing attack
k) Man-in-the-middle attack

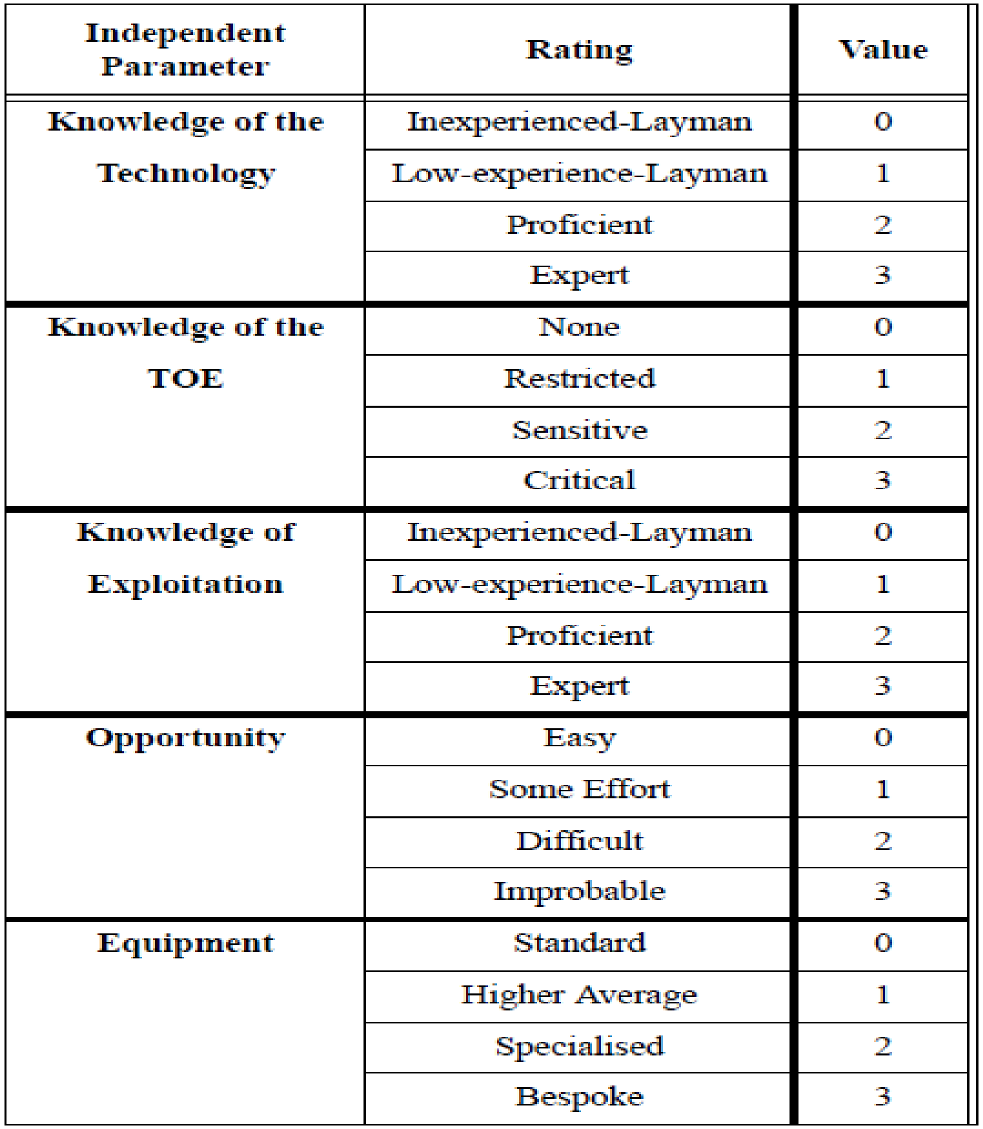
Attacks against the TCB

- bypassing
- tampering
- direct attack (by exploiting vulns in TCB)

- misused

How dangerous is an attack?

The danger of an attack decreases as the value increases

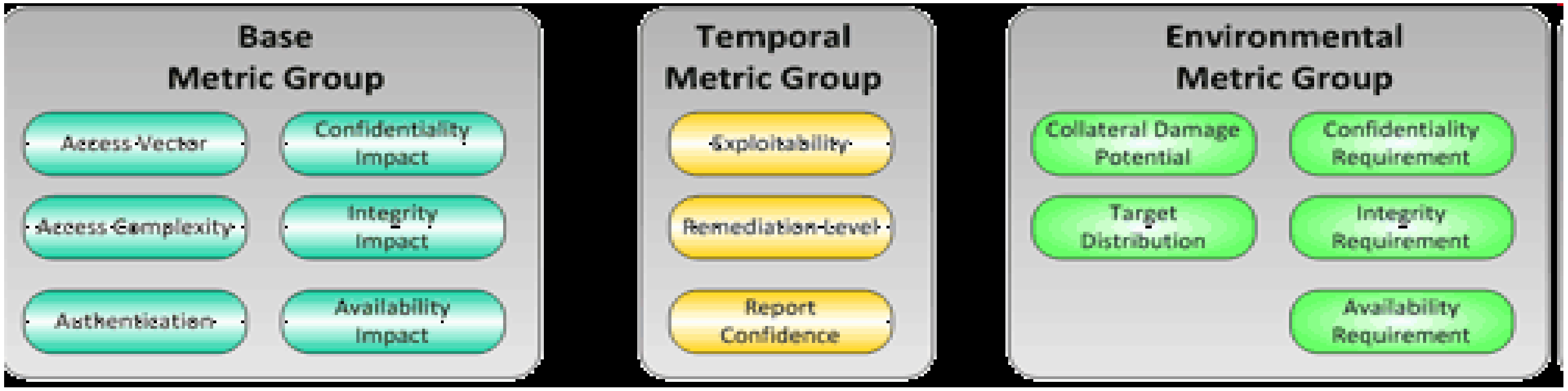


Independent Parameter	Rating	Value
Knowledge of the Technology	Inexperienced-Layman	0
	Low-experience-Layman	1
	Proficient	2
	Expert	3
Knowledge of the TOE	None	0
	Restricted	1
	Sensitive	2
	Critical	3
Knowledge of Exploitation	Inexperienced-Layman	0
	Low-experience-Layman	1
	Proficient	2
	Expert	3
Opportunity	Easy	0
	Some Effort	1
	Difficult	2
	Improbable	3
Equipment	Standard	0
	Higher Average	1
	Specialised	2
	Bespoke	3

cvss *Common Vulnerability Scoring System*

- An open framework for communicating characteristics and impacts of IT vulnerabilities in a context indipendent way
- Consists three metric groups: *Base, Temporal,* and *Environmental*
-
    - Base metric : constant over time and with user environments
    - Temporal metric : change over time but constant with user environment
    - Environmental metric : unique to user environment
    -
- Recently added the
- Authorization metrics
- Personalization metrics

CVSS (Cont'd)



CVSS metric groups

Each metric group has sub-matricies
Each metric group has a score associated with it
Score is in the range 0 to 10

Access Vector

This metric takes into account the proximity condition to exploit a vulnerability

- Local
- Adjacent Network
- Network

Access Complexity

This metric measures the complexity of the attack to exploit the vulnerability

- High: Specialized access conditions exist
- Medium: The access conditions are somewhat specialized
- Low: Specialized access conditions do not exist

Authentication

This metric measures the number of times an attacker must authenticate to a target to exploit a vulnerability

- Multiple: The attacker needs to authenticate two or more times
- Single: One instance of authentication is required
- None: No authentication is required

Confidentiality Impact

This metric measures the impact attack on confidentiality

- None: No Impact
- Partial: There is a considerable information disclosure
- Complete: There is total information disclosure
- Similar metrics for the Integrity Impact and Availability Impact

Base Score

Base Score = Function(Impact, Exploitability)

Impact = 10.41 * (1-(1-ConImp)*(1-IntImp)*(1-AvailImpact))

Exploitability = 20*AccessV*AccessComp*Authentication

Base Score Example CVE-2002-0392

- Apache Chunked Encoding Memory Corruption

BASE METRIC SCORE	EVALUATION	
Access Vector	[Network]	(1.00)
Access Complex.	[Low]	(0.71)
Authentication	[None]	(0.704)
Availability Impact	[Complete]	(0.66)

Impact = 6.9

Exploitability = 10.0

BaseScore = (7.8)

Another metrics

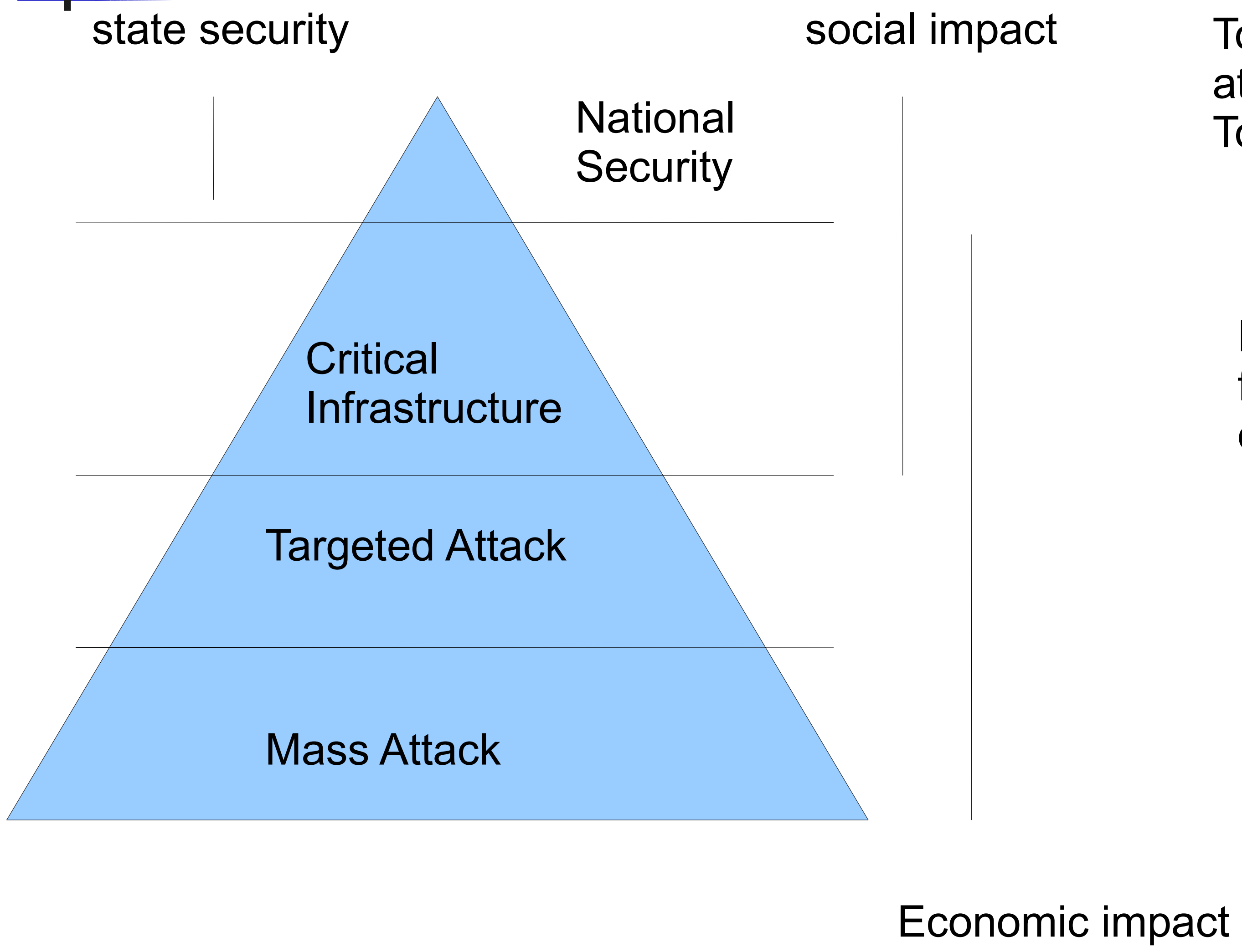
- The model assumes that the 5 coordinates are orthogonal, eg independent
- This maps each attack into a point in a 5 dimension space
    - Technology competence
    - Info on the target system
    - Attack experience
    - Probability of opportunity
    - Devices

Danger decreases with the distance from the origin of the space

A context dependent approach

- It is meaningless to evaluate how dangerous an attack may be independently of the target system
- Any evaluation should consider the pair with the attack and the system
- Let us analyze systems. ...

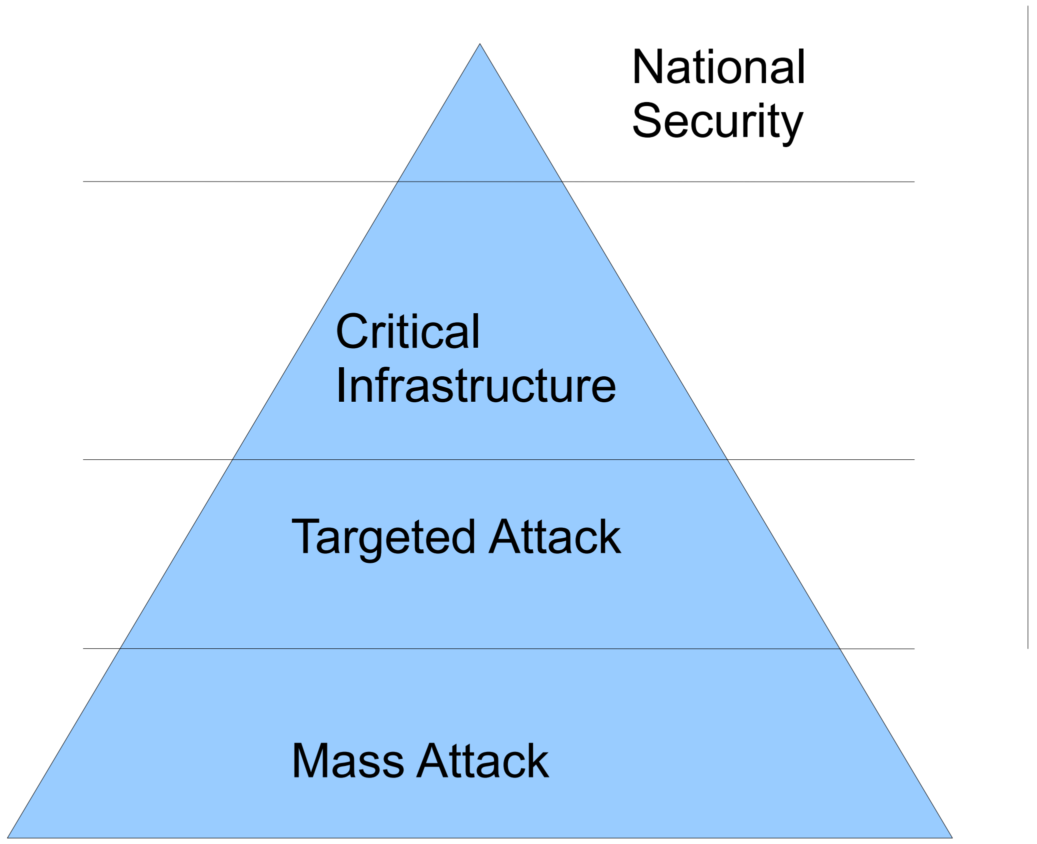
A pyramid



To understand the possible attacks first of all you have To classify your system

Higher levels also have to face the attacks of the lower ones

A pyramid



Initially we describe attacks
Against these systems

Elementary vs complex attacks

•An elementary attack is the one previoulsy described and characterized by the previous elements

•In a complex system a threat cannot achieve one goal (set of rights) through just one elementary attack

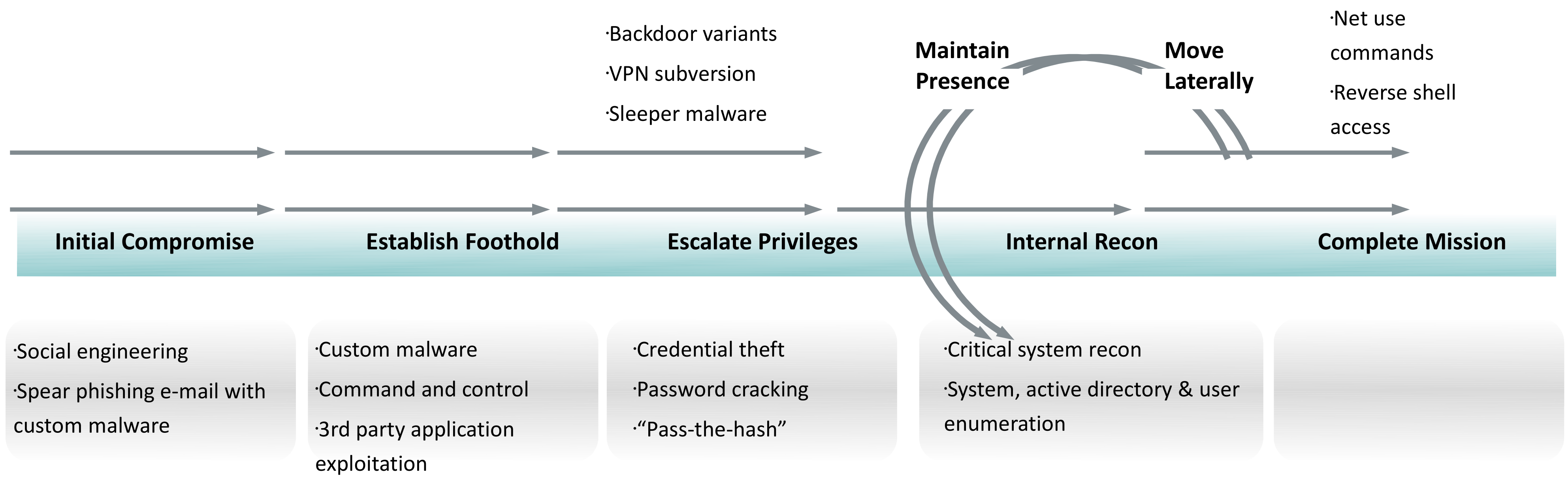
•Elementary attacks have to be composed into a complex one (attack plan, privilege escalation) to increase the rights of the attacker till reaching one of the goals of interest

•Intelligent attackers design a plan of action = an attack chain

•The precondition of each attack in the plan has to be included in the rights the attacker acquires through the previous attacks in the plan (the union of the postconditions of these attack plus any initial rights)

Complex Attack

Attackers Move Methodically to Gain Persistent & Ongoing Access to Their Targets



At organizations in the last year, the typical target attack went undetected for 273 days.

Complex Attacks - I

- Alternative points of view on a complex attack
  - Program (elementary attack = instruction)
  - Planning (steps to achieve a given goal)
  -
- Fundamental difference = coverage
  - In planning or programming we are interested in one program/strategy (optimal or suboptimal) to reach a given goal (consider one robot moving in a space)
  - Several attacks can be selected (several robots move simultaneously)
  - A risk assessment has to discover all the programs/ strategies an attacker can implement to achieve a given goal (we have to stop all the robots)

Complex attacks - II

- Elementary attacks are composed to increase the rights of the attackers (privilege escalation)
- Elementary attacks can
  - target the same system = increase the attacker rights on the system resources
  - target another system = increase the attacker rights by exploiting the trust relation among systems

Complex attack: An example

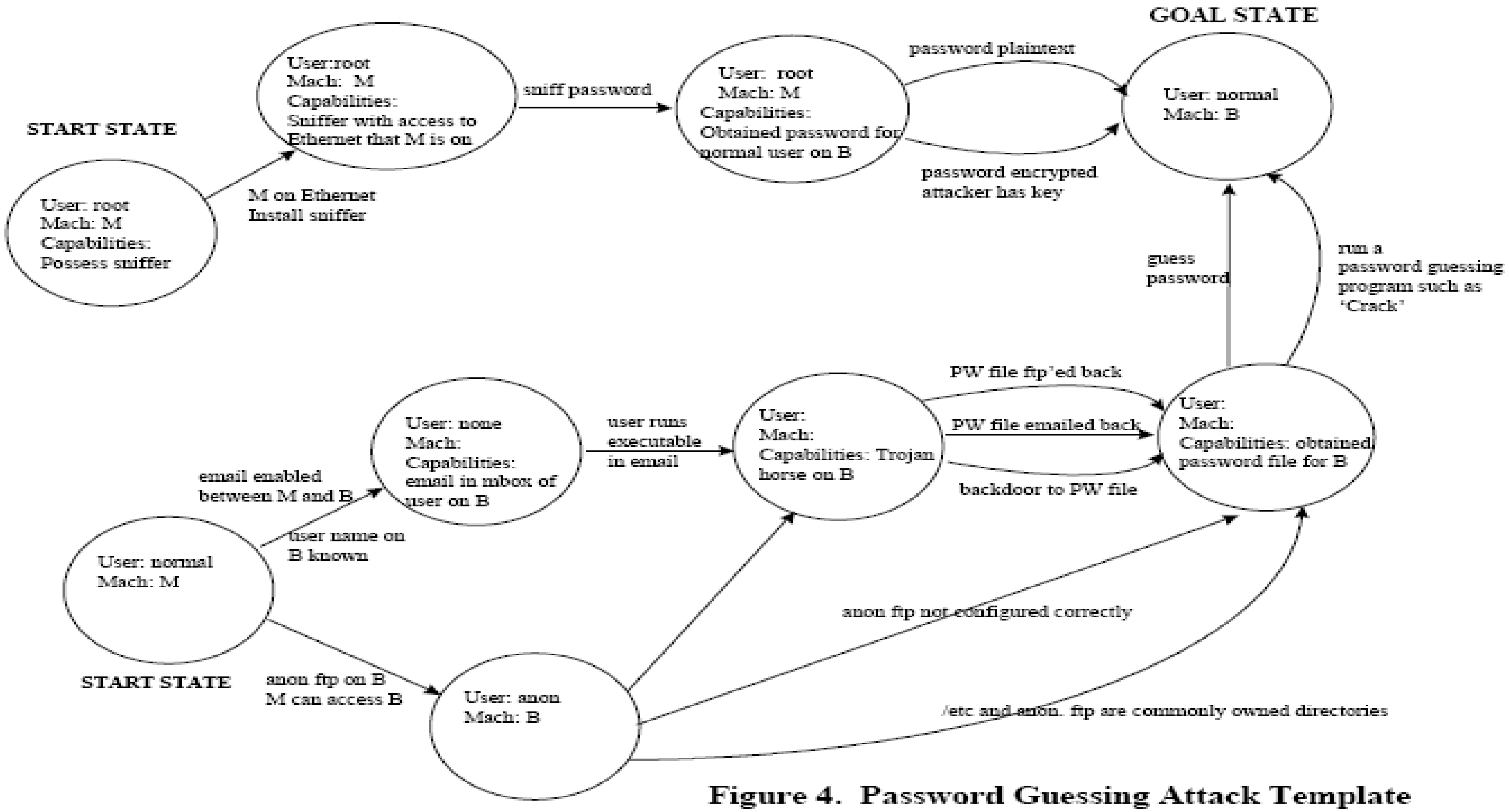


Figure 4. Password Guessing Attack Template

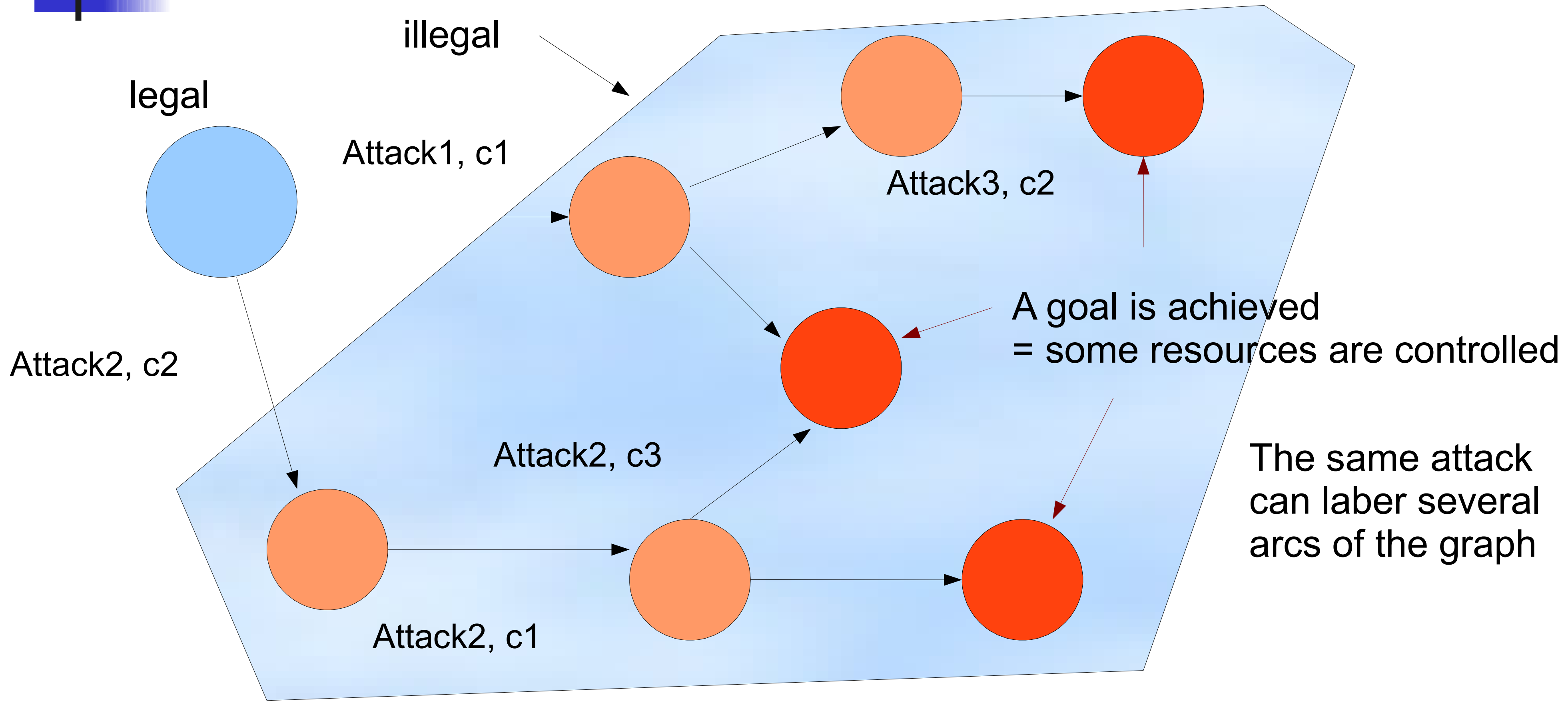
Some other example

C:\Users\X\CloudMe\didattica\1617\BHUSA09-Kortchinsky-Cloudburst-SLIDES.pdf

Attack graph

- It shows how a threat can compose elementary attacks to achieve a given goal
- Node=set of access rights
- It is a function of current vulns and of the goals of the attackers
- The graph is acyclic because of the monotone right acquisition process
- It consider the worst case where attacks are successful
- In each node the threat can execute all the attacks that are possible in the previous states – the executed one +

Evolution of a user state



Some states are useful only to reach a final state

State= set of rights

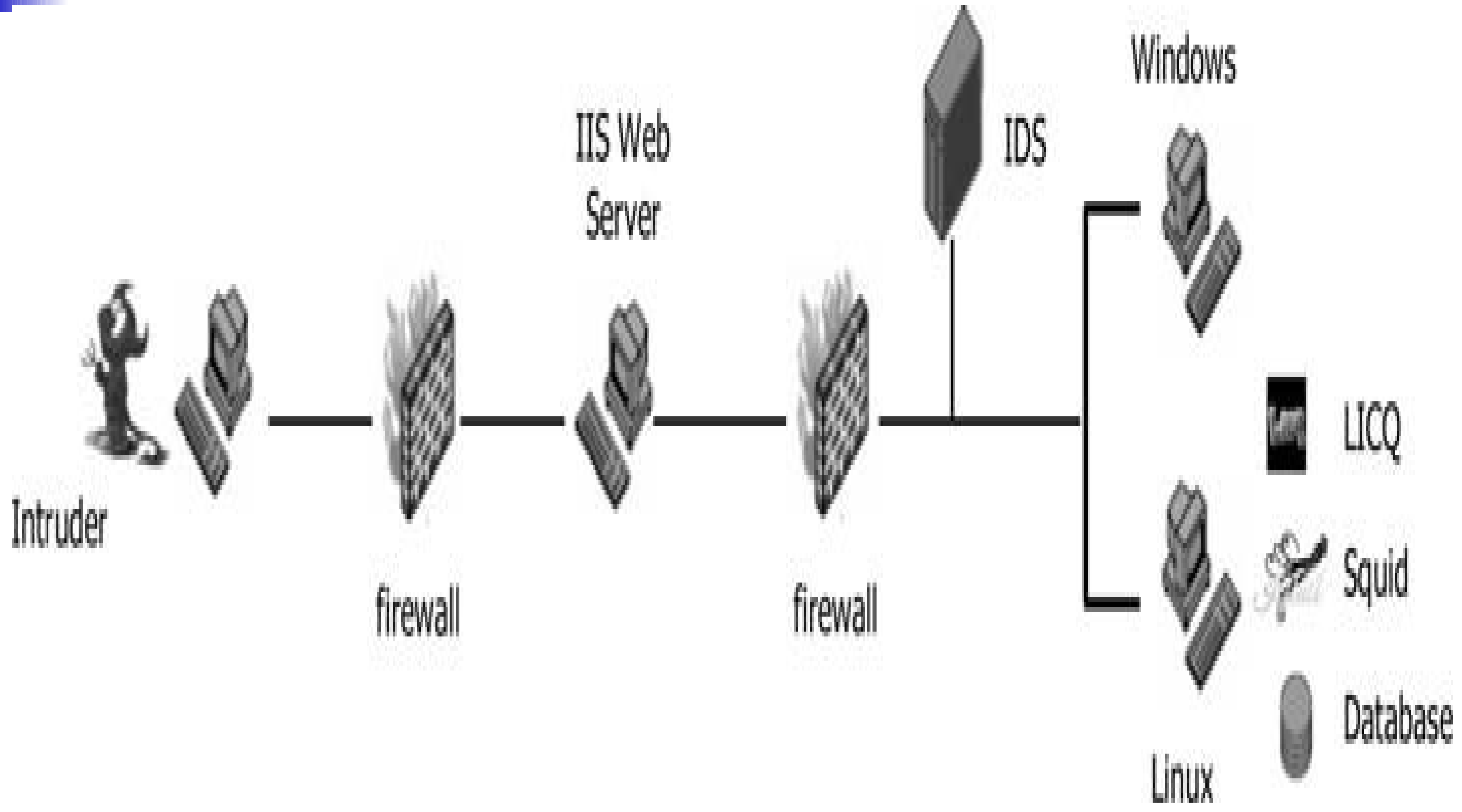
System evolution

- We can draw a graph that represents the evolution of the global system state
- The global system state is the cartesian product of the states of any attacker (user)
- Cycles are possible in the graph that describes the system evolution because a threat may reduce the rights of other ones by implementing a DOS

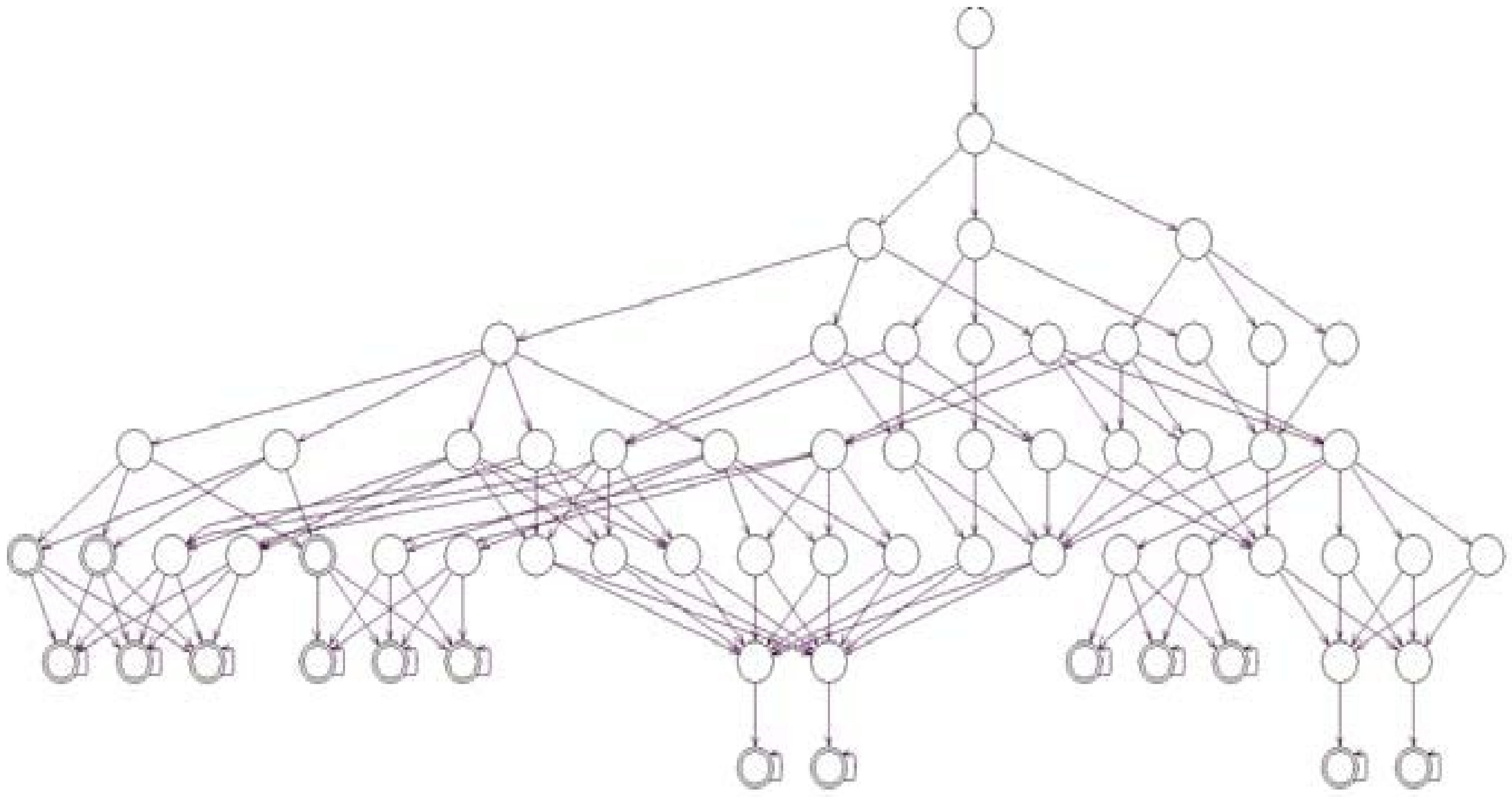
State explosion

- There is a huge number of states that strongly increases the complexity of any analysis
- It is not practical to build this graph and then analyze it due to state explosion
- Two main reasons for the explosion
  - Several attacks in a plan may commute
  - Distinct attackers can implement their attacks
    - Sequentially

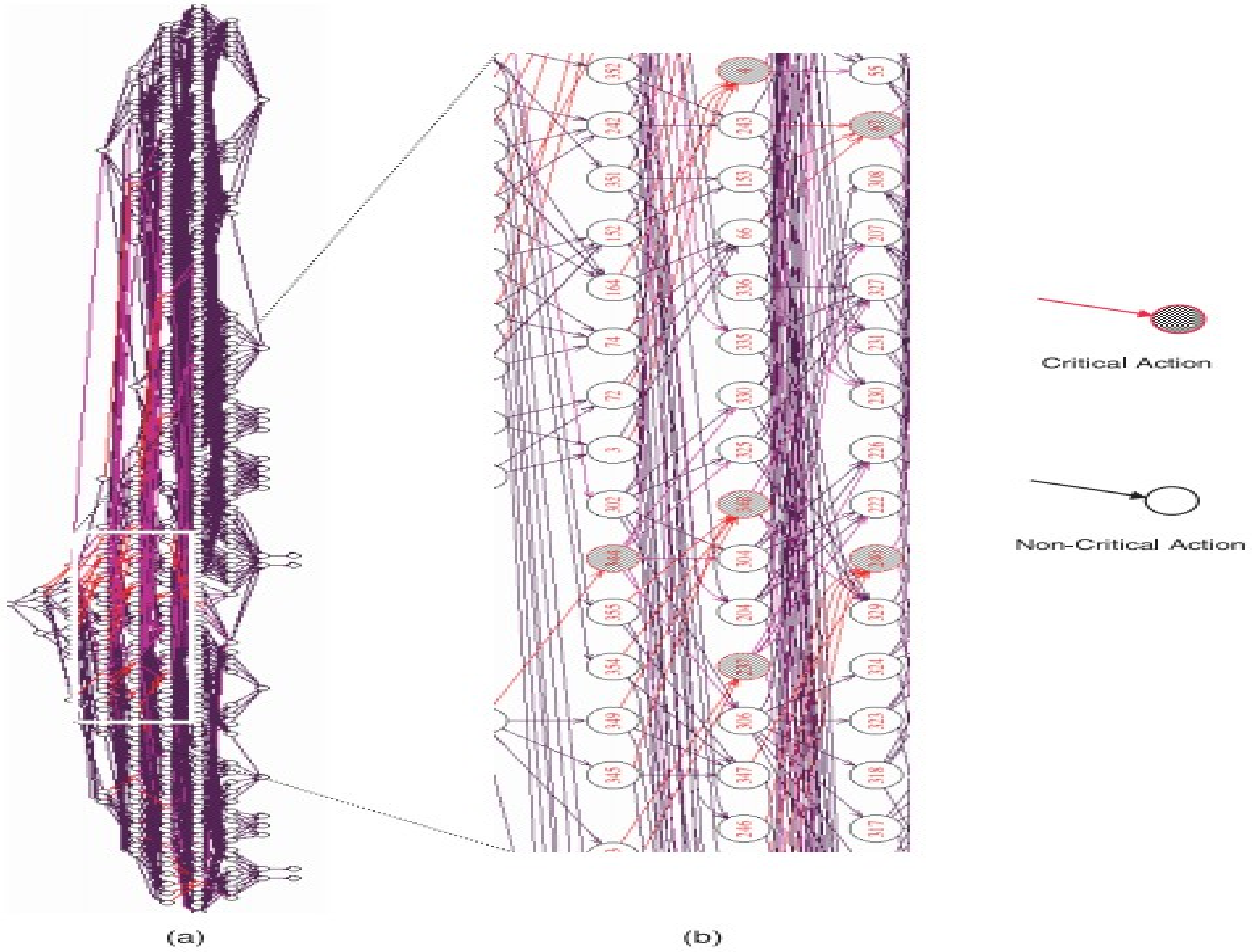
System architecture



Attack Graph



One goal of one user



(a)
(b)
Critical Action
Non-Critical Action

Monte Carlo Analysis

- The number of paths/links/nodes of the graph can be strongly reduced by focusing on an attacker behaviour
- Starting from the attack surface, we discover and build only the paths an attacker may select by simulating its behaviour according to its preferences and priorities
- Multiple executions to handle
  - Non determinism in the behaviour
  - Handling of attack failures

Monte Carlo Analysis

- The approach is based upon the joint executions of the system model and the attacker one
- Multiple joint executions build a subset of the attacker attack graph
- The accuracy of the subset depends upon the accuracy of
    - System model
    - Attacker model
    - Number of executions = confidence level

Elementary vs complex attacks

- The problem of discovering elementary attacks is rather different from discovering how the attacks are to be composed to reach a goal
- The discovery of elementary attacks depends upon both system vulns and on the components of the system that is available
- The composition of elementary attacks may be considered as an instance of a well known optimization problem = how to reach some nodes of a graph

Attack surface

•The system attack surface includes any elementary attack that is the starting points of complex attacks, the initial elementary attacks of a complex one

•An elementary attack outside the surface can be stopped by preventing the execution of some attacks in the surface

•The ratio r between the number of attacks in the surface and the overall number of attacks in attack plans may be seen as an approximated evaluation on the system security

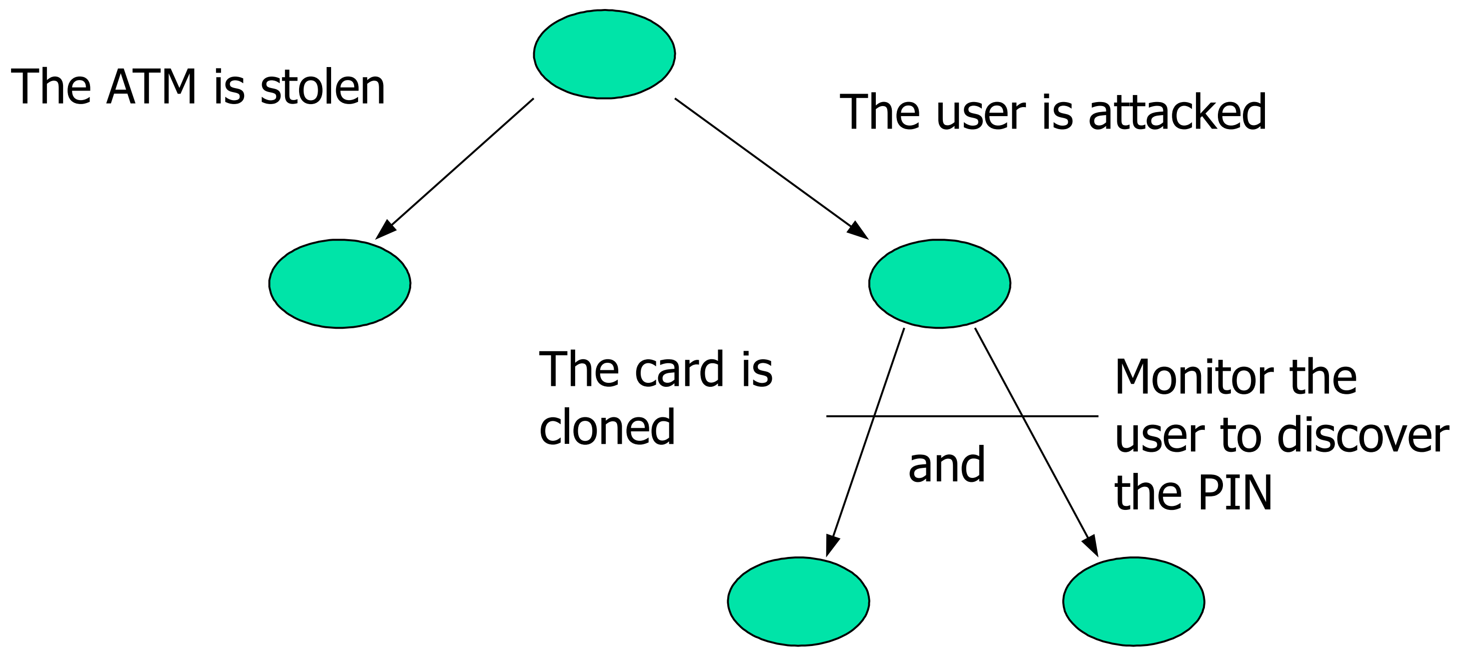
- r③1 ➢ there are several ways to compose the attacks into plans, so increasing the overall security is complex and expensive due to the large number of initial attacks
- r③0 ➢ by stopping a few attacks in the surface we stop all

Attack Tree Analysis – I

- A top down approach to discover
-
- how a complex attacks can be implemented
- How decompose a complex attack into simpler ones till we reach elementary attacks
-
- The top down decomposition ends when its frontier include elementary attacks only
-
- Two alternative decompositions
- AND = all the attacks are required
- OR = just one of the attacks is required
-

Attack Tree Analysis - II

ATM attack



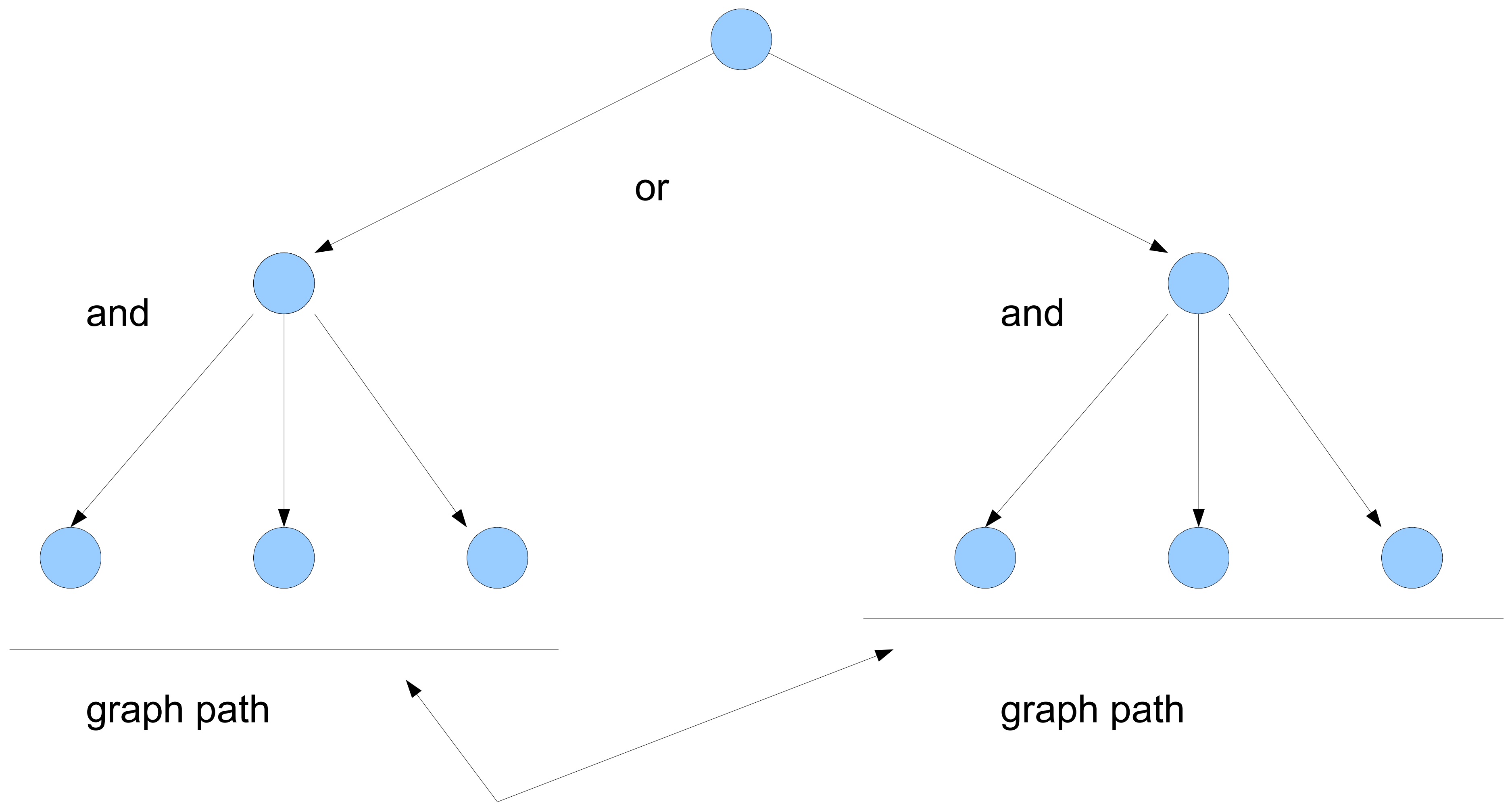
Attack Tree Analysis -III

- Thinking of a tree may be misleading because elementary attacks may be shared among subtrees
- How to discover problems shared among subtrees?
- A model based on a finite state automata may simplify the recognition of equivalent states and, hence, of common problems
- States = set of access rights that have been acquired
- Automata = attack graph

Attack tree vs graph (automata)

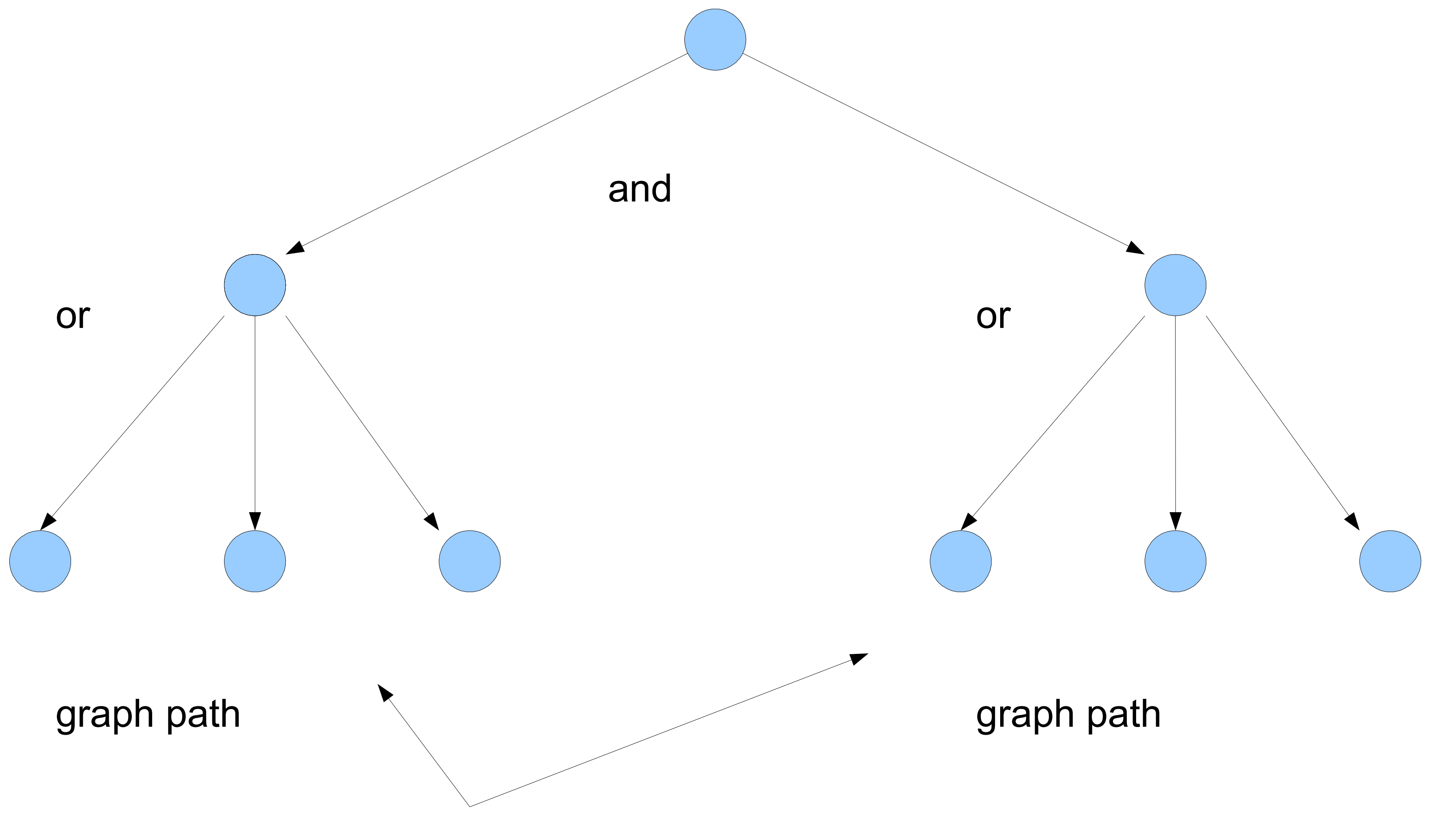
- The attacks in an AND relation in the tree belongs to the same path of the graph
- An OR nodes implies that several paths can be defined and do exist in the graph
- A tree represents one or more complex attacks
  - Consider the subtree rooted in the root of the tree
  - This subtree includes all the sons of an AND node and one son of an OR node
  - The complex attack composes all the leaves

Attack tree vs graph



Two complex attacks that are represented as two paths

Attack tree vs graph



Nine complex attacks that include one descendant of each or node

Countermeasure

- Any change to a system that decrease the success probability of an attacker
- Static countermeasure = it changes the system that is deployed for all its life
- Dynamic countermeasure = it changes the system only when it is under attack. Requires some monitoring tool to discover ongoing attacks

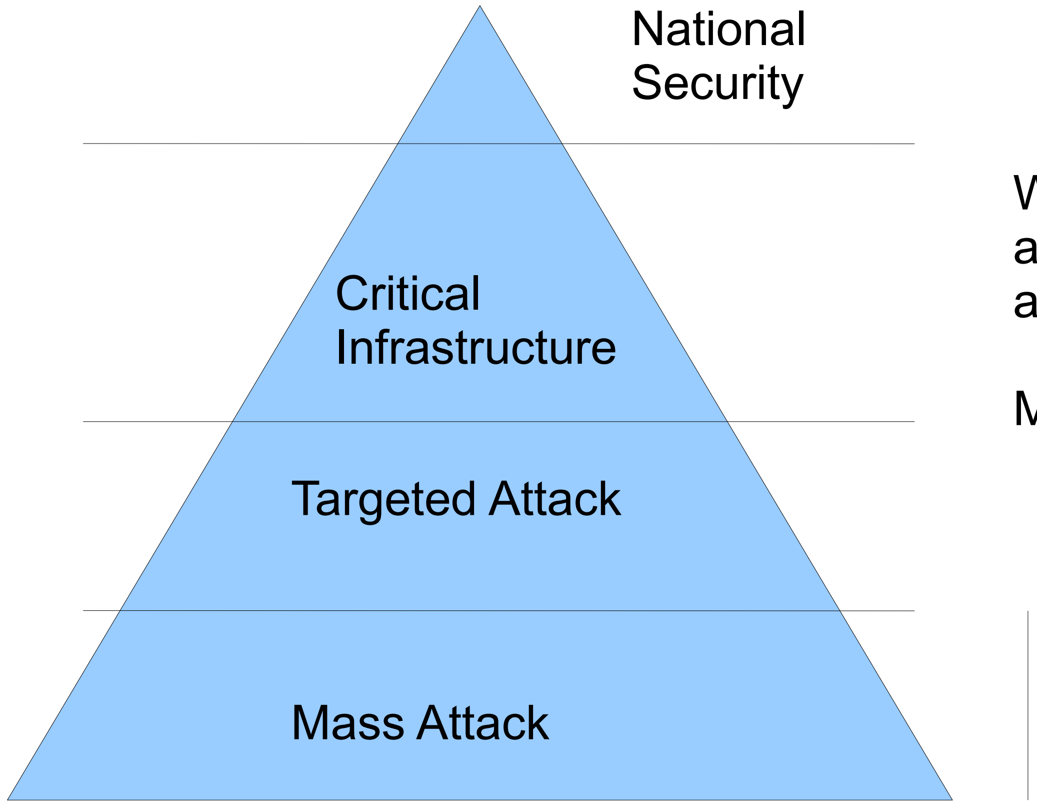
Complex attacks and countermeasures

- We stop a complex attack by stopping any of its elementary attacks
- A countermeasure of an elementary attack A stops all the complex attacks where A appears
- Cut set of an attack graph = a set of arcs (= of elementary attacks) such that no goal can be reached if they are cut (if the attacks are stopped)
- A cut set includes at least one elementary attack for each complex one that enables a threat to reach one goal (you have to discover all the complex attacks)
- Shared attacks are the key to cost effectiveness

Selecting the countermeasures

- Several cut sets may exist, each with a distinct cost
- Cost effective solutions stop
  - the most shared elementary attacks
  - attacks with cheapest countermeasures
- Betweeness = how many paths to a goal shares an arc that corresponds to the same attack

A pyramid



We consider now
attacks that can be automated
and implemented against any system

Mass Attack = Automated Attack

Risk of automatic attacks

- Original features of ICT security are
  - Fully automated attacks = fully programmable attacks
  - Automatic tools to implement attacks (execute the program)
- The existence of tools that implement the attacks
  - Simplify the implementation of attacks
  - Strongly enlarge the pool of potential attackes

The risk strongly depends upon the feasibility of automating an attack

Fully automated attacks

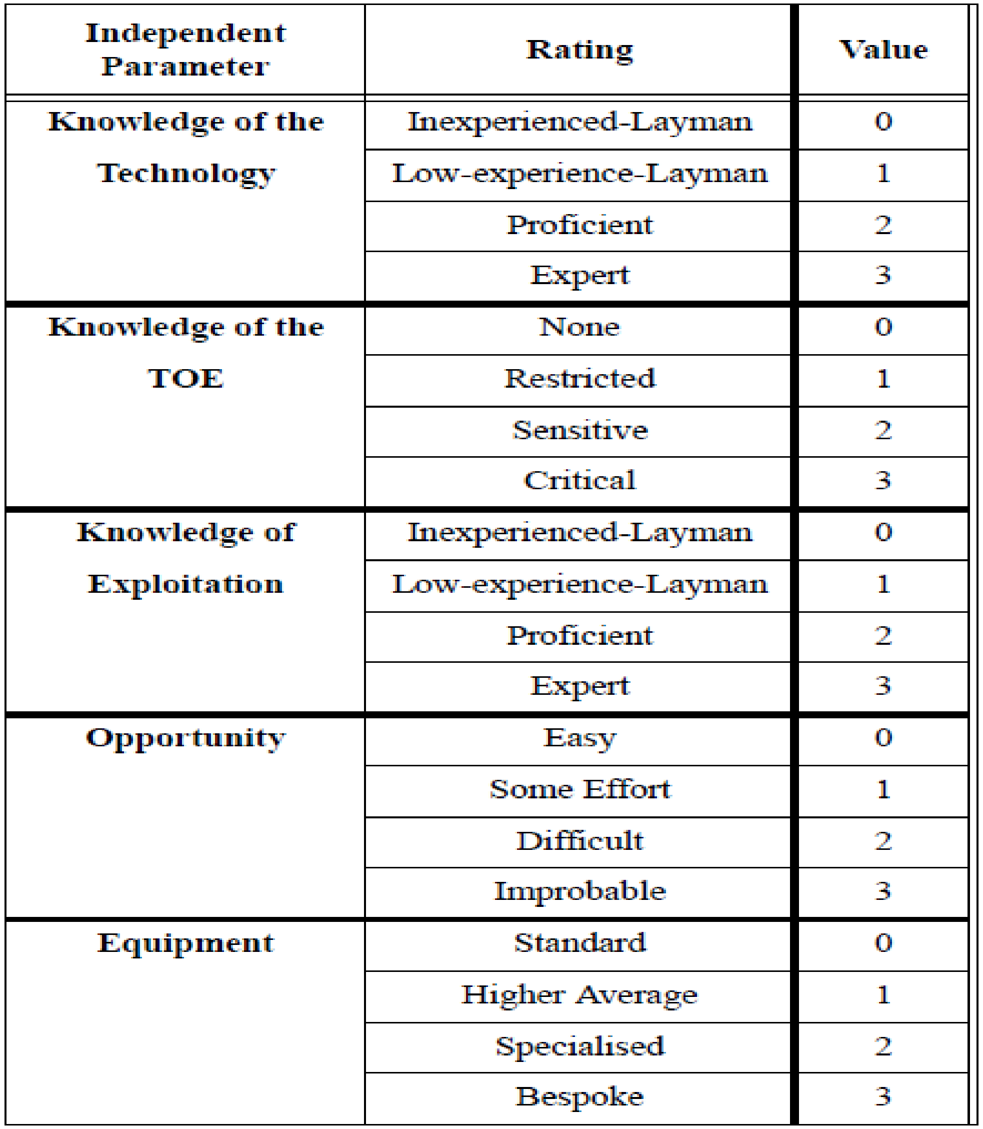
- Exploit = the program that exploit the vulnerability to implement the attack to control some components
  = executed against the instances of a standard comp.
- All the instances of a standard component
  - Are affected by the same vulns
  - Can be attacked by the same exploit
- Fully automated attack= no further actions, information, abilities are required besides the ability of running the exploit
- In the dangerous evaluation that applies five dimensions, the first 3 are equal to zero and the fifth one is outside the control of the defender
- Currently, several exploit databases are available that store exploit that can be tested against a system

Fully automated remote attacks

- A fully automated attack that can be launched from another node
- The attack grants an account on the target node
- These attacks are the starting point to write code that
- replicates itself on an attacked node
- A threat can write a worm using the exploits of these attacks
- A worm can sequentially execute any number of exploits

How dangerous is an attack?

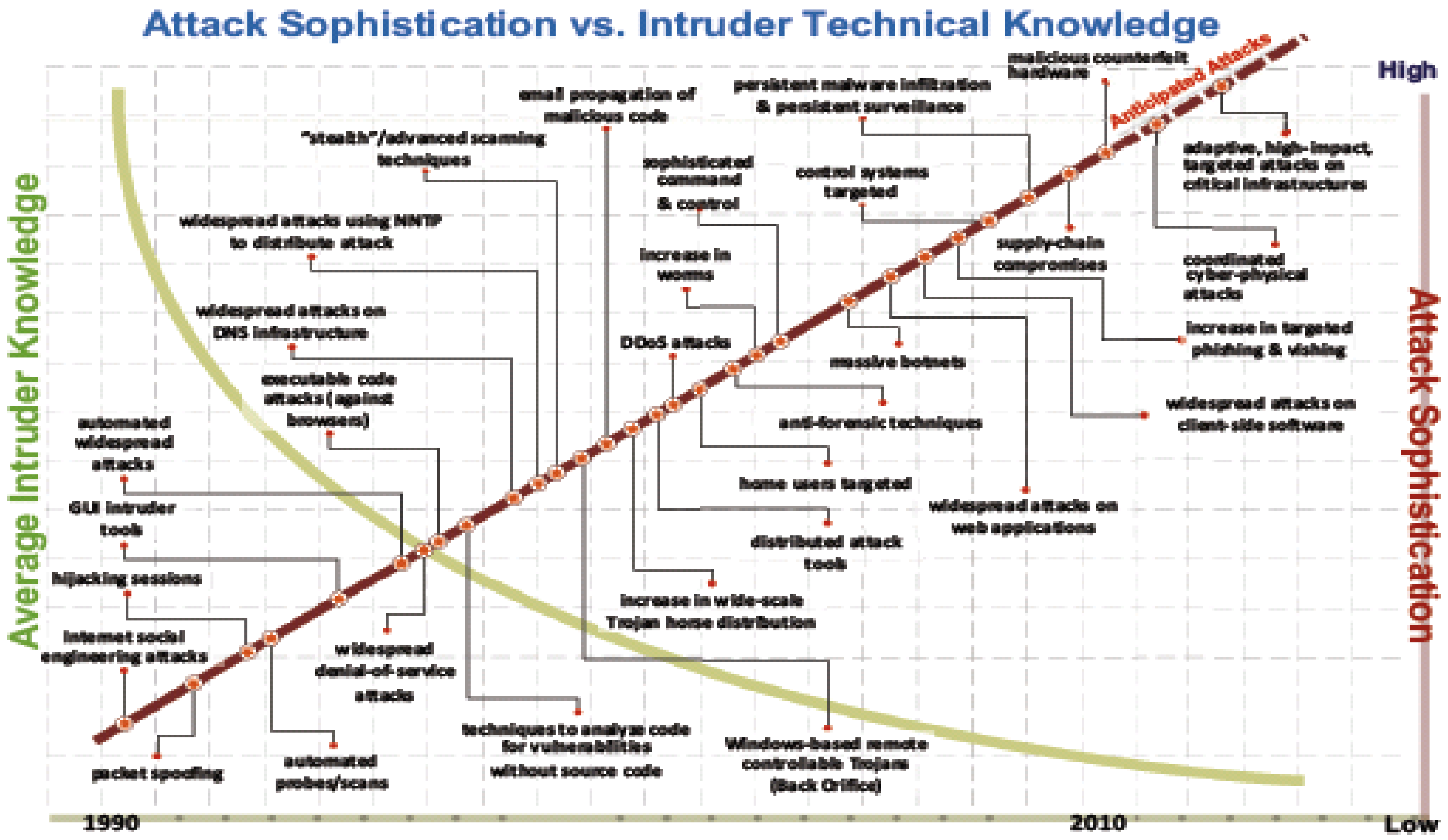
The danger of an attack decreases as the value increases



Independent Parameter	Rating	Value
Knowledge of the Technology	Inexperienced-Layman	0
	Low-experience-Layman	1
	Proficient	2
	Expert	3
Knowledge of the TOE	None	0
	Restricted	1
	Sensitive	2
	Critical	3
Knowledge of Exploitation	Inexperienced-Layman	0
	Low-experience-Layman	1
	Proficient	2
	Expert	3
Opportunity	Easy	0
	Some Effort	1
	Difficult	2
	Improbable	3
Equipment	Standard	0
	Higher Average	1
	Specialised	2
	Bespoke	3

All zero if fully Automated attack

Fully automated attacks



inf

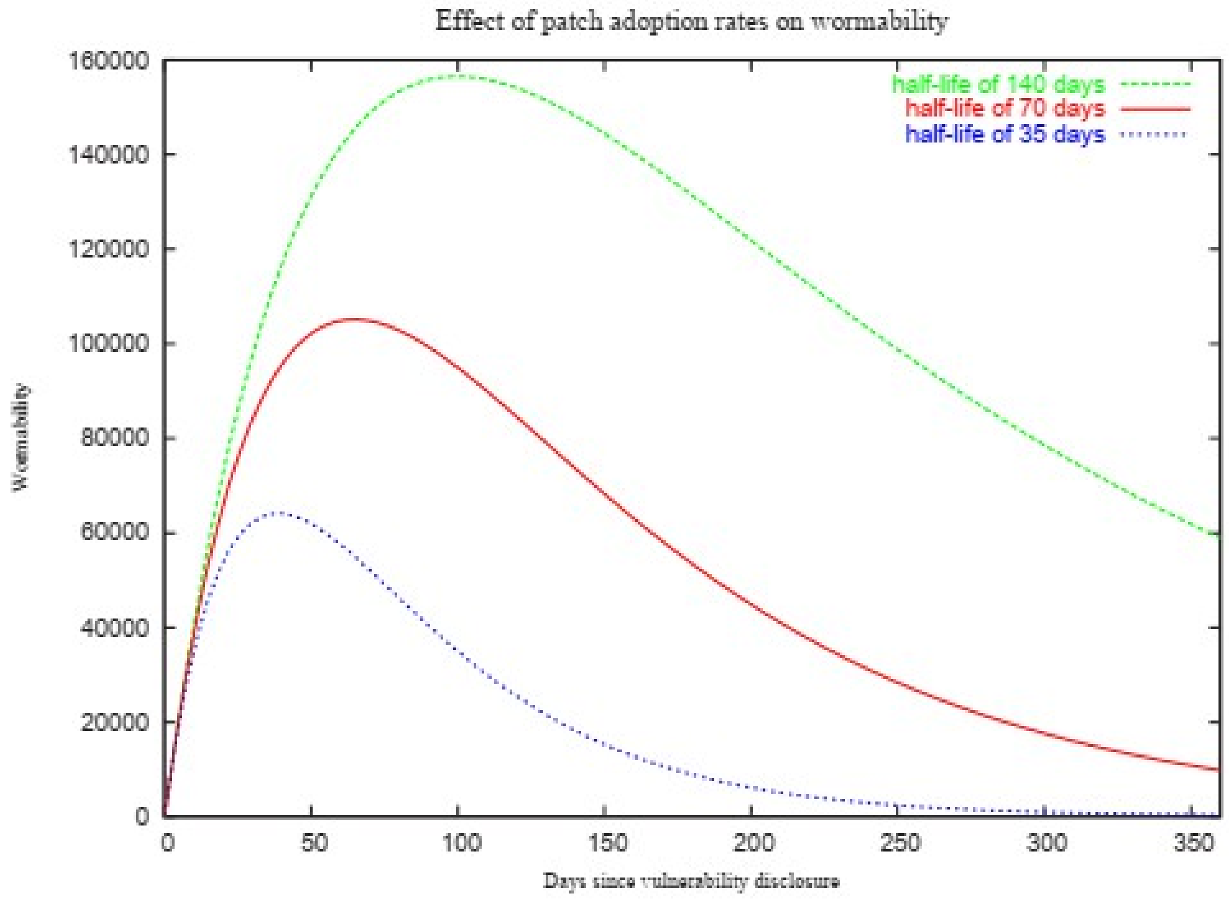
Fully automated attacks

- The functions show how really dangerous attacks may be implemented through tools that are distributed and accessed through the web
- It is more and more critical the window of exposure = the time interval between
  - The time an exploit is pubblicly available
  - The vuln is removed from the system

  ➢ even a complex organization has to apply the patches to remove a vuln in a very short time

(good point to remember with the next slide)

Patch adoption



Fully automated attacks: an example

```
Thu Feb 24 09:45:47    HTTP request from 202.109.114.209: POST /_vti_bin/_vti_aut/fp30reg.dll
        Thu Feb 24 09:45:54    possible overflow attempt via HTTP from 202.109.114.209 (request line is 65552 bytes long)
        Thu Feb 24 09:45:54    HTTP bogus request from 202.109.114.209: SEARCH
        /□HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
        HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH...

Thu Feb 24 15:48:21    possible overflow attempt via HTTP from 81.30.200.55 (request line is 65552 bytes long)
        Thu Feb 24 15:48:21    HTTP bogus request from 81.30.200.55: SEARCH
        /□HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
        HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH...
        Thu Feb 24 15:48:23    HTTP request from 81.30.200.55: POST /_vti_bin/_vti_aut/fp30reg.dll

Thu Feb 24 15:57:37    possible overflow attempt via HTTP from 218.43.229.149 (request line is 65552 bytes long)
        Thu Feb 24 15:57:37    HTTP bogus request from 218.43.229.149: SEARCH
        /□HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
        HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH...
        Thu Feb 24 15:57:41    HTTP request from 218.43.229.149: POST /_vti_bin/_vti_aut/fp30reg.dll

Thu Feb 24 16:00:34    HTTP request from 61.54.219.101: GET /default.ida?
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
%u9090%u685...
```

Three attacks in two seconds

The ICT zoo (malware)

- Virus
- Worm
- Trojan Horse
- Hybrid
- Autonomous Hybrid

Most important problem

In the future

Ransomware Attack Impacts Aluminum Production

https://www.nozominetworks.com/blog/breaking-resear

•According to media reports, the malware attack began on the evening of Monday, March 18th, Oslo time (UTC + 1). On March 19th, the company's website was not available and production impacts had been reported:

•Potlines, which monitor molten aluminum, and need to be kept running 24 hours a day, had been switched to manual mode

•Some factories have been forced to halt production

•Several metal extrusion plants have been closed

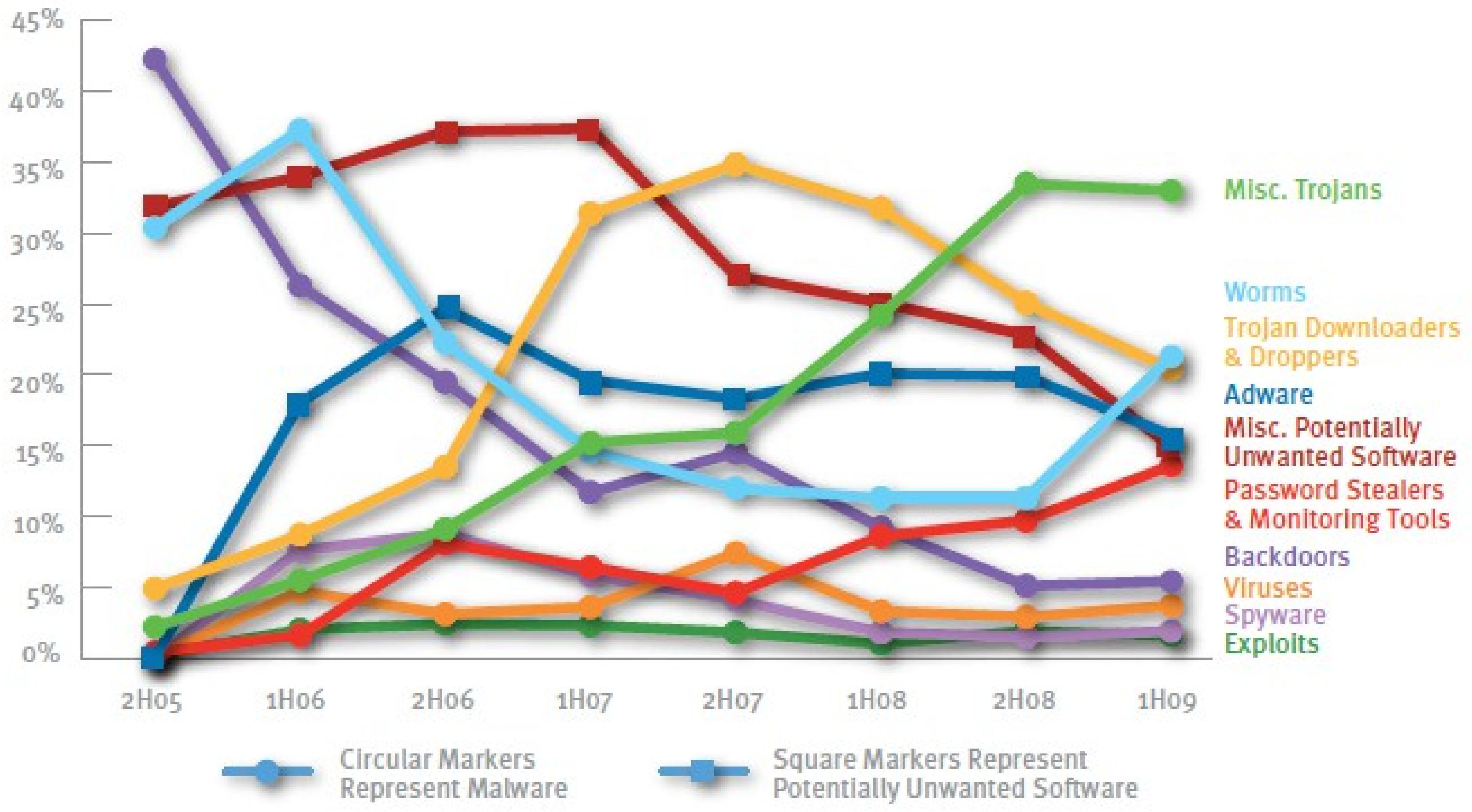
•At certain facilities, some computer systems are unavailable, and printed orders are being fulfilled

•Power plants are functioning normally

•No safety-related incidents have been reported

Some statistics

FIGURE 10. Computers cleaned by threat category, in percentages, 2H05–1H09



Virus

- A program that
- Hides itself in other program or data
- It is transmitted together with such a program or such data (parasite)
- Can be activated at a prefined time
- The behaviour is fully dependent upon the programmer of the virus
- Currently USB keys and devices are the main diffusion mechanisms

Fully automated and mobile attacks

- Worms and virus implement automated attacks and can replicate onto attacked nodes
- Worm=a program that after successfully attacking another node, creates a copy of itself onto this node
  - Attack vector = the code to attack (infect) other nodes
  - A payload (send spam, steal/update/modify node info)
  - Connect to a C&C and download the payload
  - Domain flux
- The worm attacks any node the infected one can reach
- Genetic diversity is important but multiple versions of a

Command&Control

•Some nodes under the control of the worm writer

•They can update the worm attack vector and payload

•Domain flux = generation of alternative domains nodes or aliases for these nodes to increase the complexity of a shut down (detection mechanism)

•Botnet= overlay network including the nodes that have been attacked and controlled by the worm creator rather than by the legal owner

Sapphire/Slammer worm

- 376 byte in one UDP packet
- It exploits a vuln in the SQL server
- An infected node can infect from 100 to 10000 further node in one second
-
- The number of infected nodes (worm metric doubles in 8.5 seconds
- ➏ 100 times faster than previous worms
- More than 75.000 infected nodes

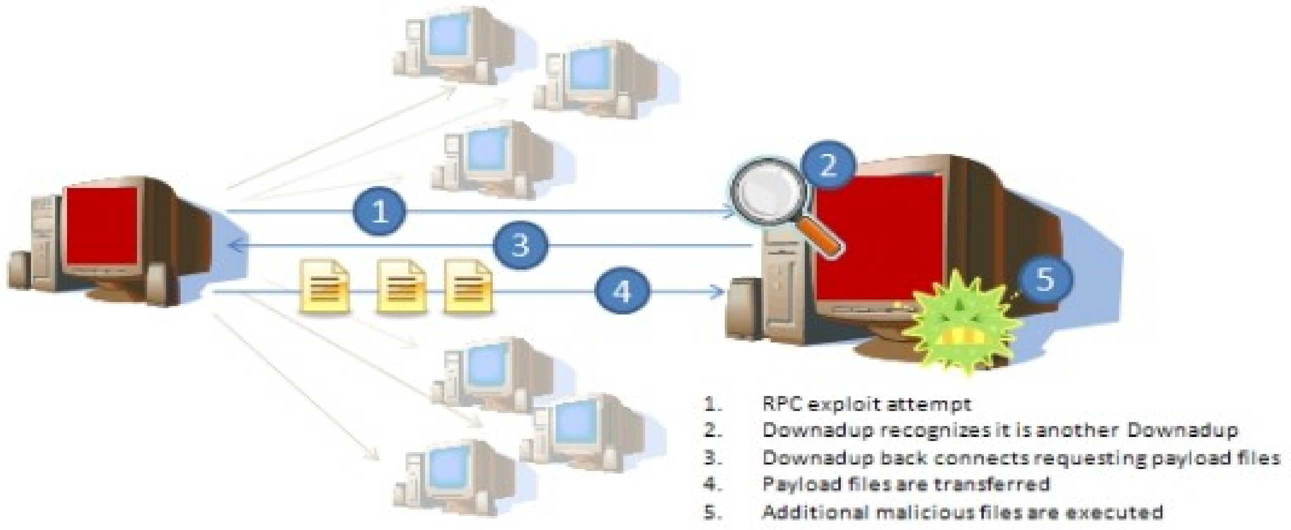
Sapphire/Slammer worm …

- In 10 minutes it has infected 90% of nodes that may have been infected =
worm attacks will be successful
- Not sure this is a “good” feature
- It creates a lot of “noise” that strongly simplifies attack detection
- “Stealth worm” = slow attack, low amount of noise, difficult detection

Conficker: an hybrid

- Can attack:
- Windows 2000, Windows XP, Windows Vista, Windows Server 2003, Windows Server 2008, e Windows Server 2008 R2 Beta
- Hybrid as it can exploit: USB device, share and email
- 9 milions system attacked (e.g. English defence dept, french air army, hospitals) in jan. 2009
- 30% of nodes is currently vulnerable
- It can download updates, 5 versions

Conficker vs p2p


1
2
3
4
5
1. RPC exploit attempt
2. Downadup recognizes it is another Downadup
3. Downadup back connects requesting payload files
4. Payload files are transferred
5. Additional malicious files are executed

•Let us assume that an infected node is attacked again

•The infected node

- understands that the attacker is a peer (is infected)

Conficker

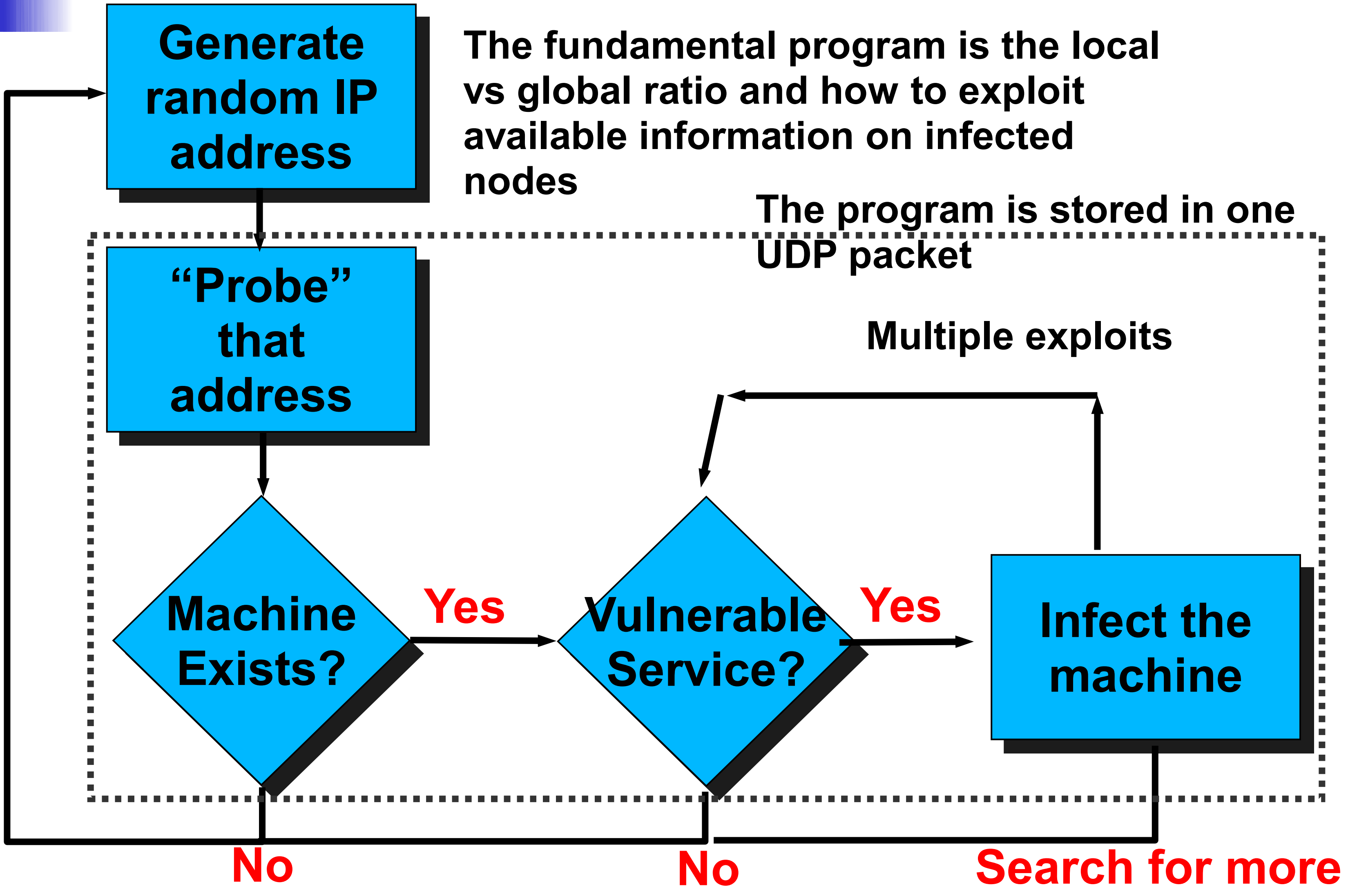
- It implements Domain flux to download the updates
- Input/output connessions are encrypted
- Payload = information collection + creation of a botnet
-

An important point

“Whereas a missile comes with a return address, a computer virus (or worm) generally does not.”

Deterrence and Dissuasion in Cyberspace, J.Ney

The general structure of a worm



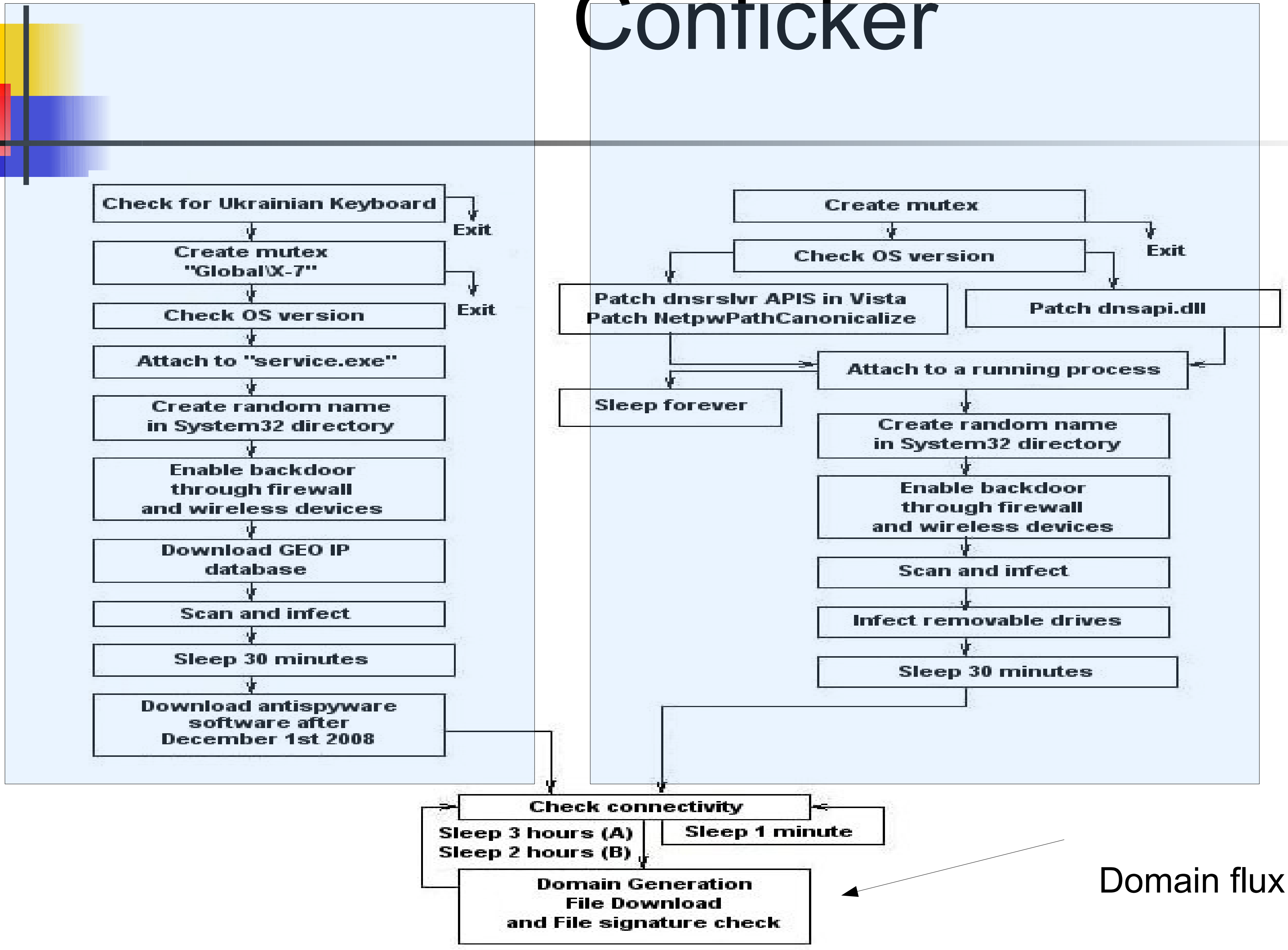
The fundamental program is the local vs global ratio and how to exploit available information on infected nodes

The program is stored in one UDP packet

Version A

Version B

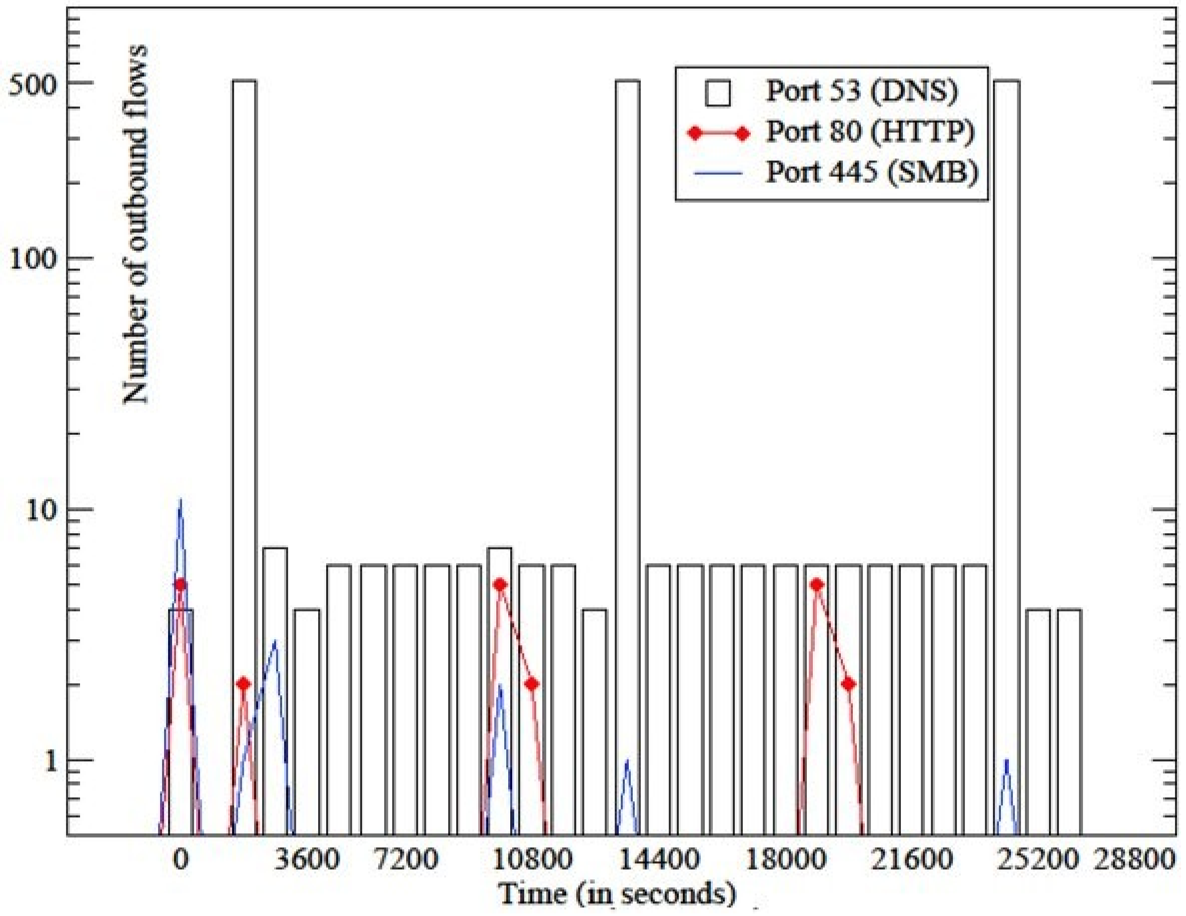
Conficker



Check for Ukrainian Keyboard
Exit
Create mutex "Global\X-7"
Exit
Check OS version
Attach to "service.exe"
Create random name in System32 directory
Enable backdoor through firewall and wireless devices
Download GEO IP database
Scan and infect
Sleep 30 minutes
Download antispyware software after December 1st 2008
Create mutex
Exit
Check OS version
Patch dnsrslvr APIS in Vista
Patch NetpwPathCanonicalize
Patch dnsapi.dll
Attach to a running process
Sleep forever
Create random name in System32 directory
Enable backdoor through firewall and wireless devices
Scan and infect
Infect removable drives
Sleep 30 minutes
Check connectivity
Sleep 3 hours (A)
Sleep 2 hours (B)
Sleep 1 minute
Domain Generation
File Download
and File signature check

Domain flux

Conficker

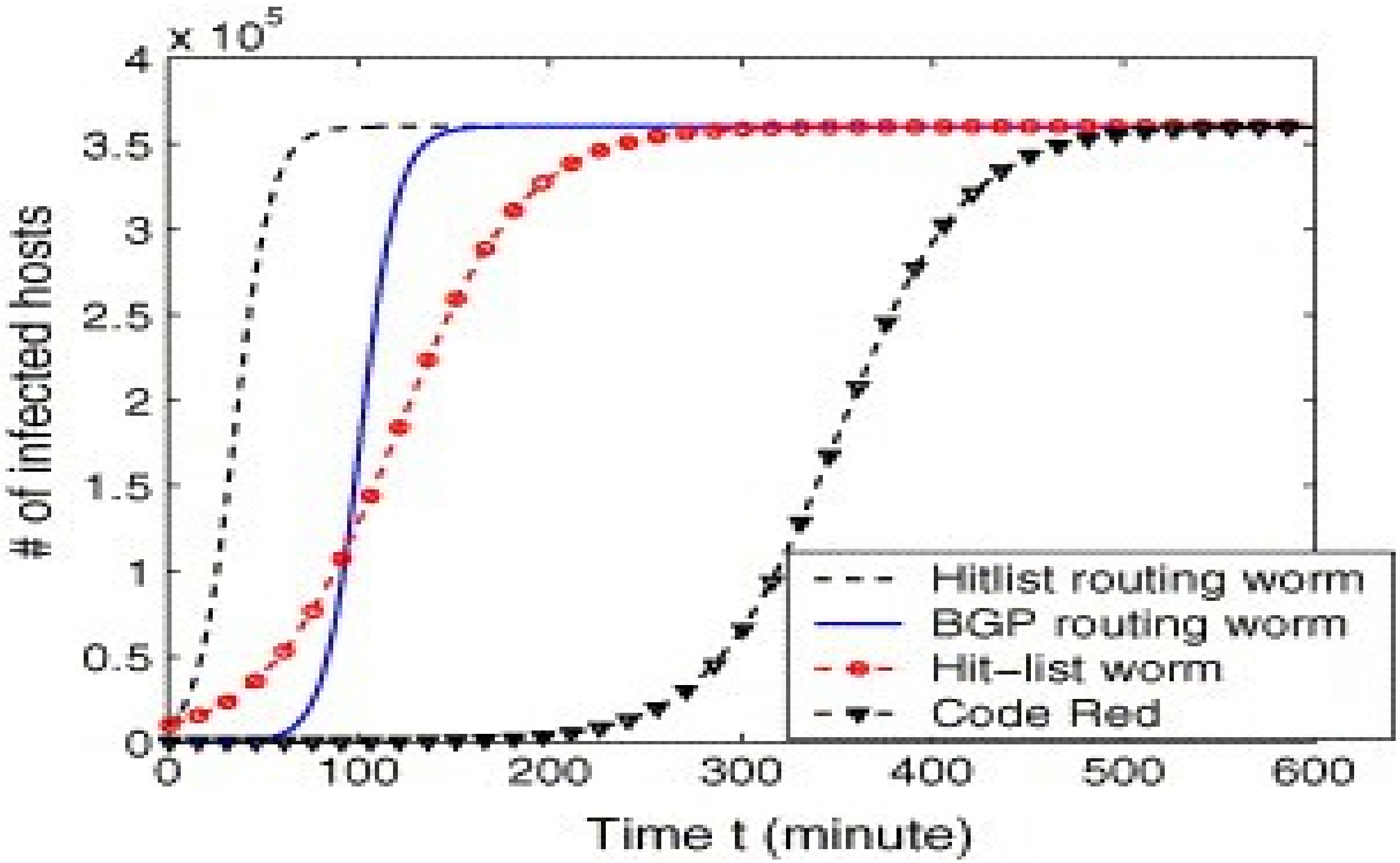


Generation of IP addresses in an infected nodes

Address generation

- Two disjoint subsets
- Local (high density) = subnet of the infected node
- Global (low density)
- Density = the probability that a random address belonging to the set corresponds to a real node
- If the ratio of local vs global addresses is too low the worm may be detected and removed before spreading, eg infecting other nodes
- If the percentage is too large, then after infecting all nodes resources are wasted because one node may be infected several times
- Even low changes in the ratio may be very critical, non linear effects

The influence of the ratio



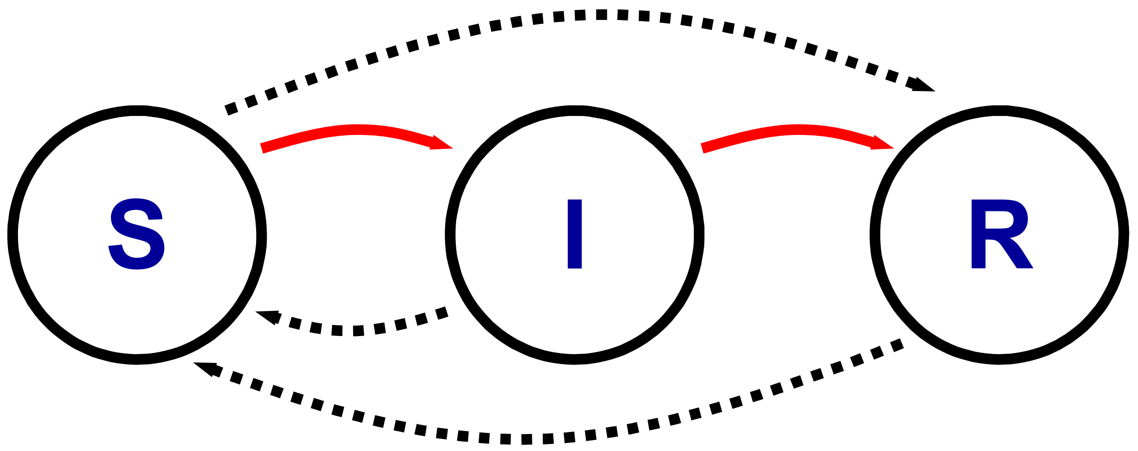
A detection strategy

- Some proposals aim to detect infected nodes by the anomalous behavior resulting from the random generation of addresses
- High rate of failed connections
- Two thresholds can be introduced
    - Distance on the scanned hosts
    - Frequency of the scanning
- Further features to discover worms
    - New host contacted
    - Unused addresses used

A theoretical spreading model

- Let us discuss a theoretical model to study the spreading of a worm
- The model is epidemiological = it has been defined to evaluate the number of people infected overtime
    - because of a contagious illness
    - in a closed population
    - fully connected population

A finite state model of individual to study the spreading



Model states

•susceptible = Host that may be infected

•Infected = Infected host

•Recovered = Host that cannot be infected

Typical transition sequences (red arrows)

•The host runs the software that is vulnerable (potential).

•The worm has exploited the vuln and successfully attacked the node (infected).

•The infection is detected and the system reconfigured (recovered).

A set of diff equations

Classic epidemiology

- [Kermack and McKendrick, 1927]
- Alll the nodes follows the red paths in the automata (P tlo I, I to R)

$$\frac{\mathrm{d}s}{\mathrm{d}t} = -\beta s i$$

$$\frac{\mathrm{d}i}{\mathrm{d}t} = \beta s i - \gamma i$$

$$\frac{\mathrm{d}r}{\mathrm{d}t} = \gamma i$$

s = potentially infected

i = infected

r = recovered

Beta = infection rate

Gamma = recovery rate

Gamma may be neglected in the case of worms because the time to spread is very litte

Kermack and McKendrick model

- ❂ is a function of
  - The function to generate the IP addresses
  - The number of the system affected by the vulns
- It increase with the virulence
- The model assume that a node can infected any other node =
  complete connection and no defence
- ❊ should not be neglected anytime

Epidemiological threshold

R_0= ❂s /❊

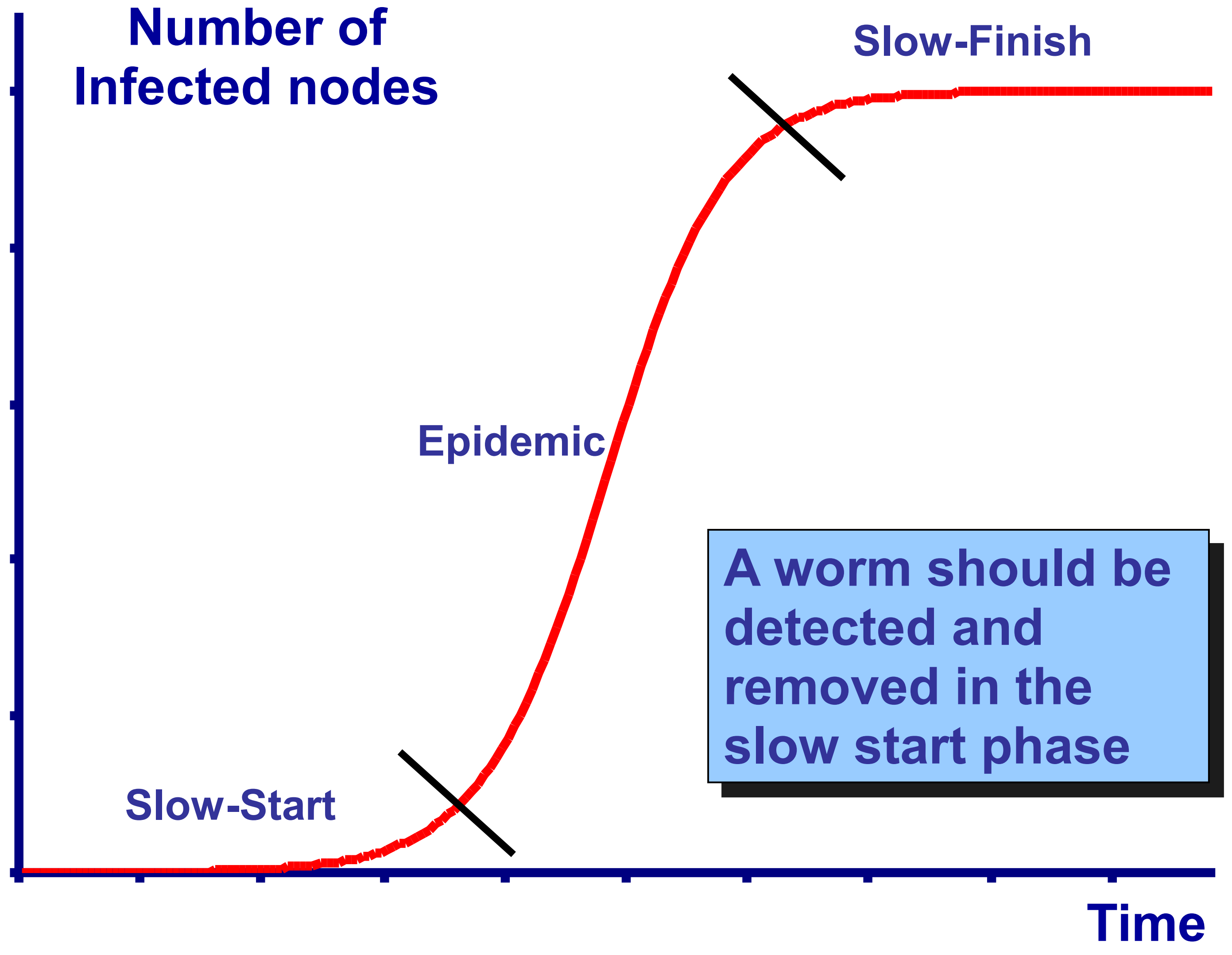
- s= percentage of nodes that may be infected
- It is the average number of nodes infected by an infected node
- If R_0 ✞ 1 the worm spreads, otherwise it will be defeated

Solution of the system of diff equations

- No exact solution can be computed
- Anytime the initial number of infected may be neglected (I(0)✠0) then

$$I(t+\Delta t)=I(t)+pcS(t)\frac{I(t)}{N}\Delta t-I(t)\gamma\Delta t+o(\Delta t).$$

Solution = logistic function



A worm should be detected and removed in the slow start phase

A model that consider patching

dS(t)/dt = -❂ S(t)I(t) - dP(t)/dt

dR(t)/dt = ❊I(t)

dP(t)/dt = ❍S(t)I(t)

dI(t)/dt = +❂ S(t)I(t)

S(t) + I(t) + R(t) + P(t) = N

patched

There are two reasons why a node is no longer susceptible

1. It has been infected
2. It has been patched
3.

The number of patched nodes is proportional to the susceptible and of infected ones

A more complex model

(H1) The nodes outside the network are all susceptible.

(H2) The nodes outside the network are connected to the network at constant rate $\mu > 0$.

(H3) Every node in the network is disconnected from the network with constant probability per unit time $\delta > 0$. Clearly, we have $\frac{\mu}{\delta} = N$.

(H4) Due to connections with infected nodes, at time t every susceptible node in the network gets infected with probability per unit time $\beta_1 I(t)$, where $\beta_1 > 0$ is a constant. This hypothesis captures the distributed nature of virus propagation.

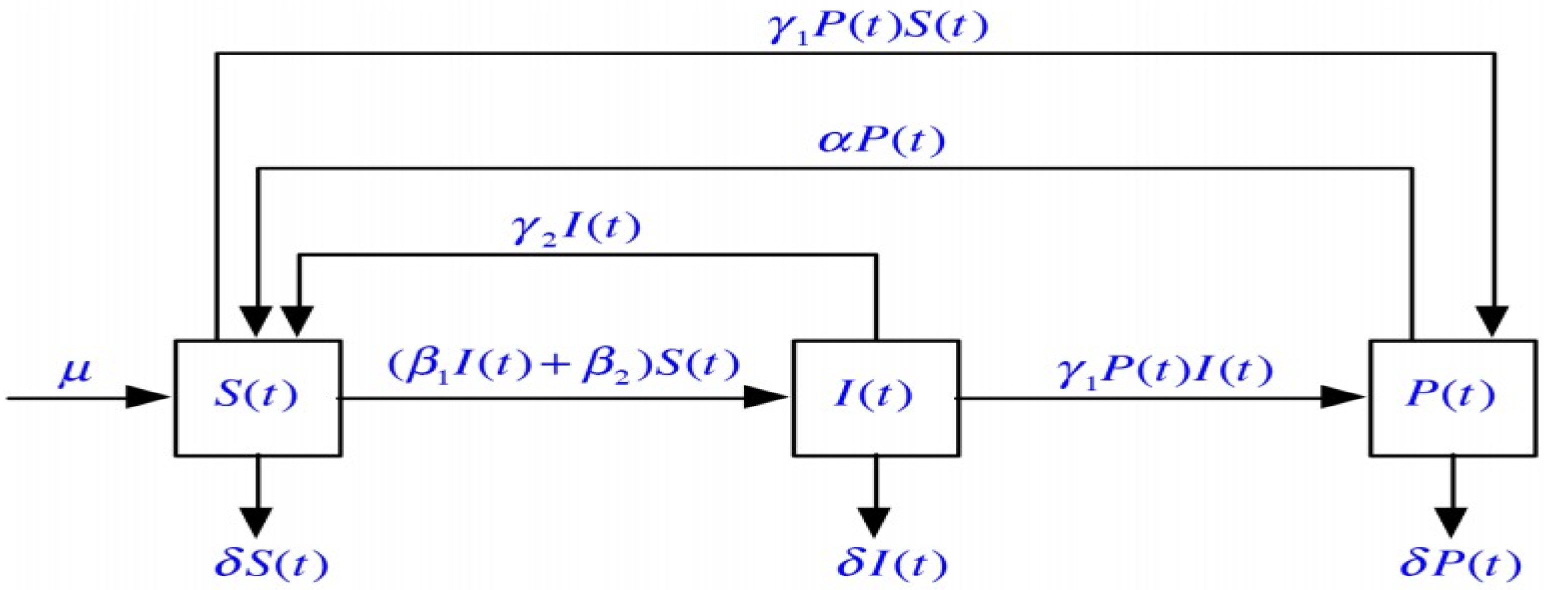
(H5) Due to existence of infected removable storage media, every susceptible node in the network gets infected with constant probability per unit time $\beta_2 > 0$.

(H6) Due to connections with patched nodes, at time t every susceptible or infected node in the network acquires the newest patch with probability per unit time $\gamma_1 P(t)$, where $\gamma_1 > 0$ is a constant. This hypothesis captures the distributed nature of patch dissemination.

(H7) Due to system reinstallation, every infected node in the internet becomes susceptible with constant probability per unit time $\gamma_2 > 0$.

(H8) Due to patch invalidation, every patched node in the network becomes susceptible with constant probability per unit time $\alpha > 0$.

A more complex model - II



$$\begin{cases} \dfrac{dS(t)}{dt} = \mu - \beta_1 S(t)I(t) - \beta_2 S(t) - \gamma_1 S(t)P(t) + \gamma_2 I(t) + \alpha P(t) - \delta S(t), \\ \dfrac{dI(t)}{dt} = \beta_1 S(t)I(t) + \beta_2 S(t) - \gamma_1 I(t)P(t) - \gamma_2 I(t) - \delta I(t), \\ \dfrac{dP(t)}{dt} = \gamma_1 S(t)P(t) + \gamma_1 I(t)P(t) - \alpha P(t) - \delta P(t), \end{cases}$$

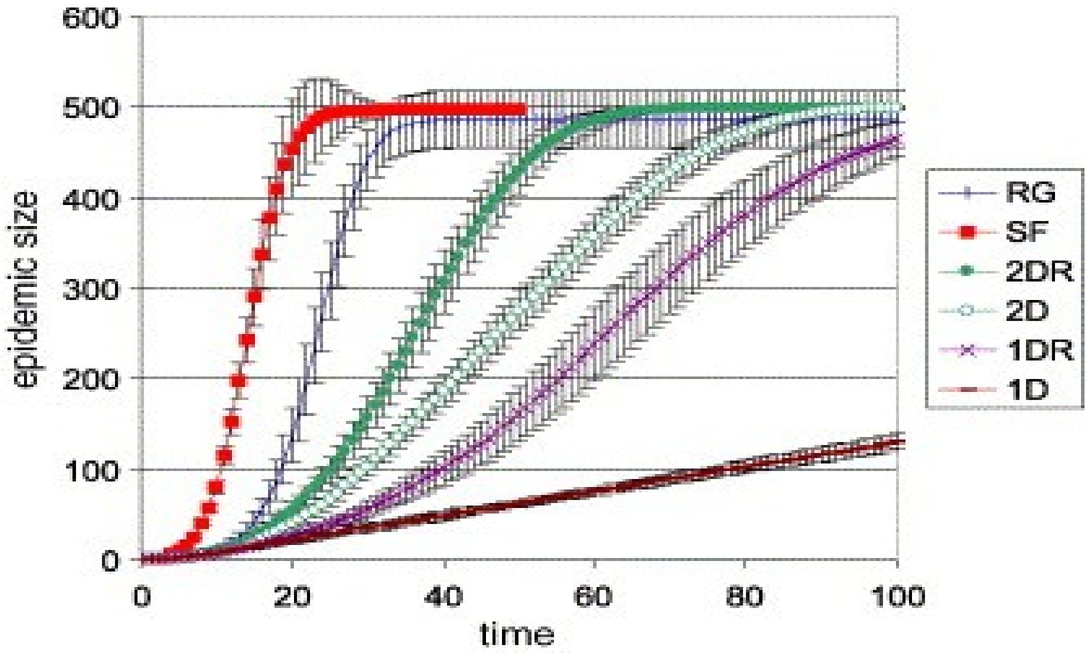
Further interesting models

- Let suppose that there is a partial connection among nodes (scale free, small world, …)
- Initially some nodes are infected
- We would like to know
    - How the connection structure influences the spreading and the parameter R_0
- How patching (=vaccination) influences the spreading
- Alternative vaccination strategies
- Several topologies may be be considered to discover how they influence the spreading

Scale free

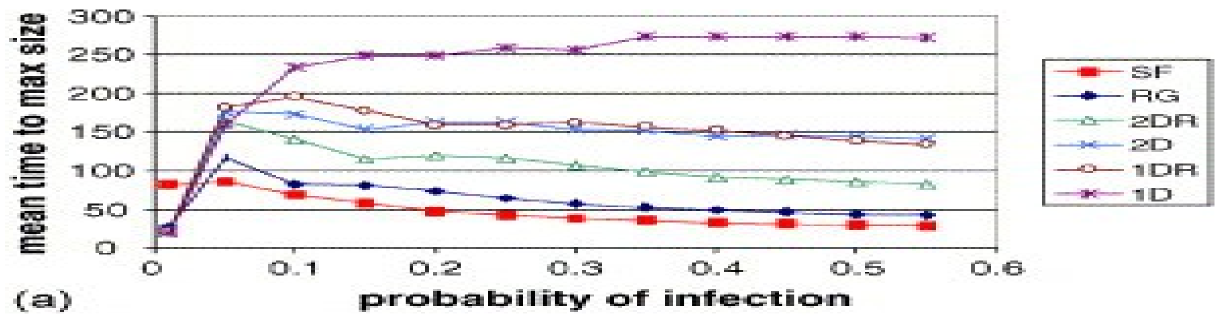
- Scale free
    - When a connection is created, nodes with a larger number of connections are preferred
    - The rich becomes richer
    - There are some network hubs with an exponential increase in the number of their connections
- Very robust with respect to random node attacks, highly fragile with respect to intelligent attacks

Interconnection Topology

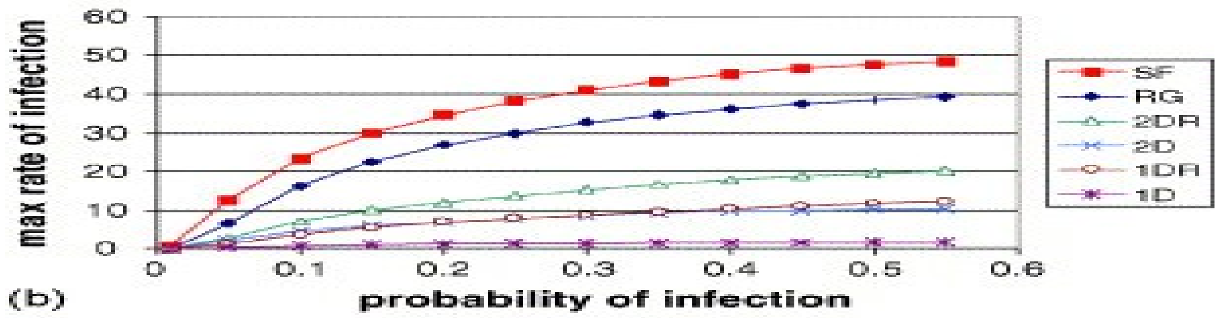


RG=random, SF=scale free, 2D= two dimensions lattice,
1D= one dimension lattice 2DR= two dimensions lattice rewired ,
1DR= one dimension rewired

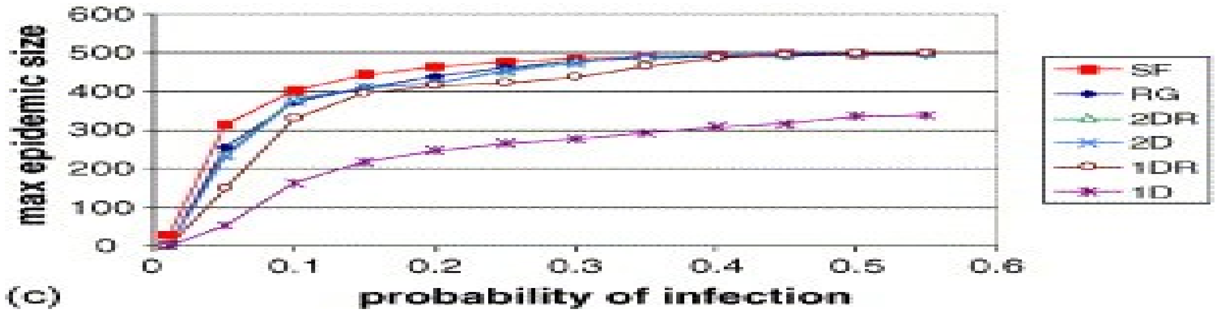
Other interesting values



Average time to max infected



Max infection rate



Number of infected

Computing a worm β

$$\beta = \frac{C}{N} \times \frac{\overset{\textit{Alpha}}{\alpha}}{\underset{\textit{Tau}}{\tau}}$$

C = 1 (a random machine is selected)

C= N (an infected machine is always selected)

N = 2^{32} (size of IP address)

Alpha = number of nodes tested in parallel

Tau =average time for testing a machine

Code red

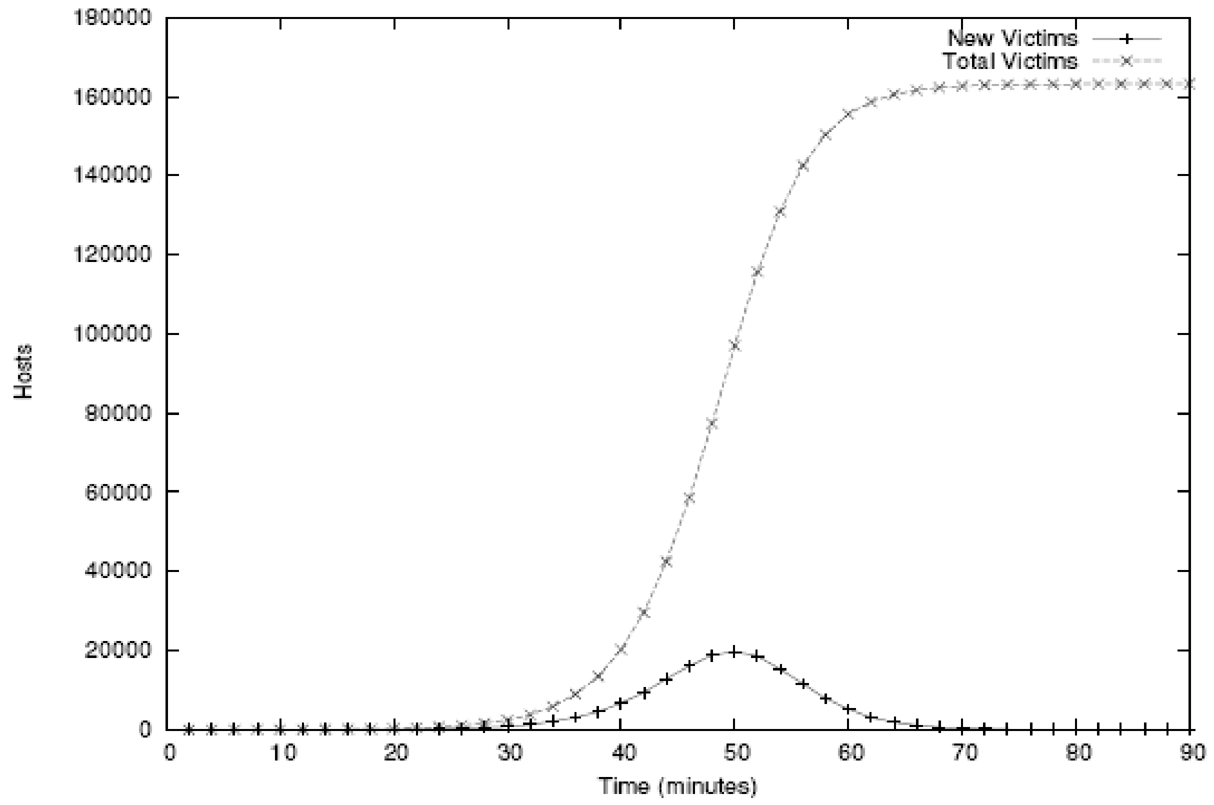
***Tau* = 19 seconds**

***Alpha* = 100**

$$\beta = \frac{1}{2^{32}} \times \frac{100}{19} = 1.23 \times 10^{-9}$$

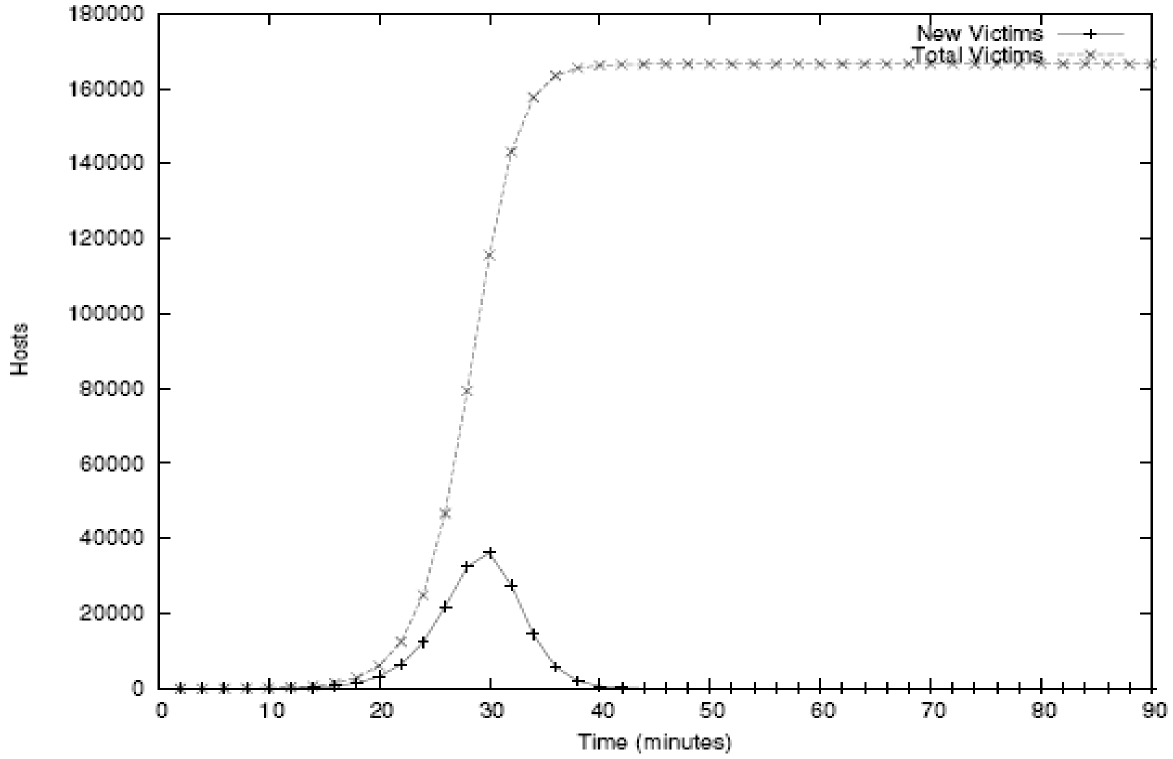
Good approximation

Spreading - I



10 parallel threads and conflicts on nodes to be infected are neglected

Spreading - II



Optimization of the time out to detect that no node has the IP address that has been generated

Spreading - III

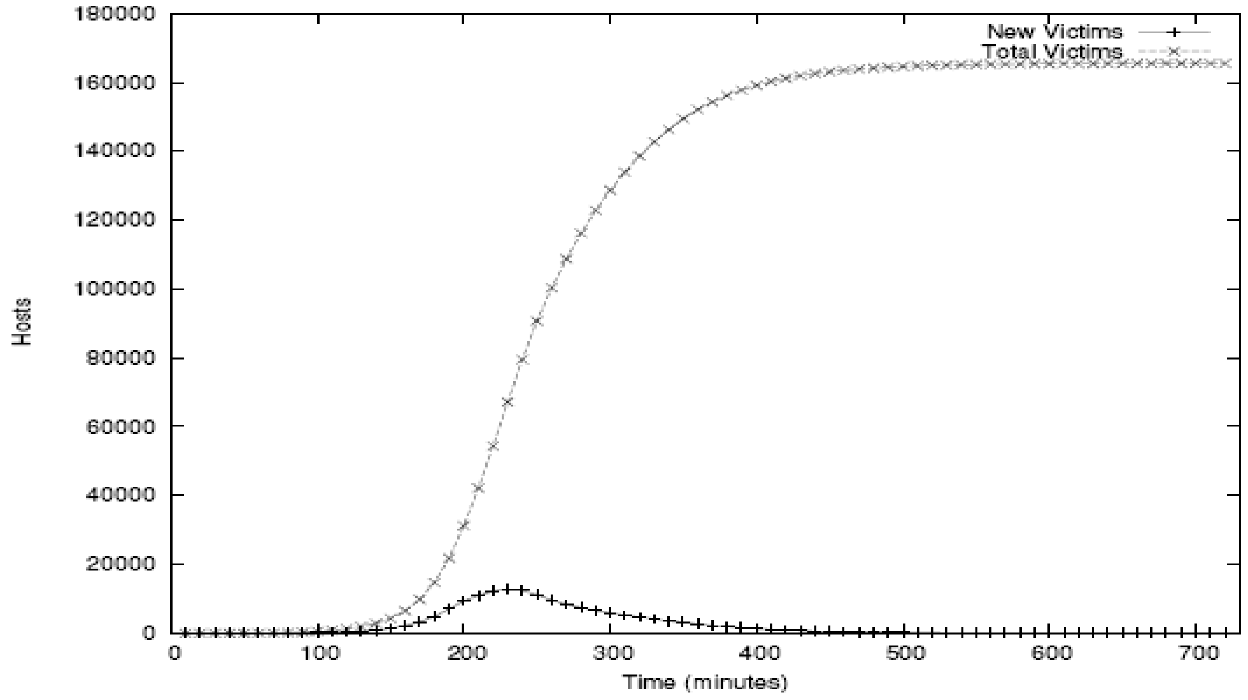


Figure 7: Local preference propagation

Local bias in the generation

Spreading - IV

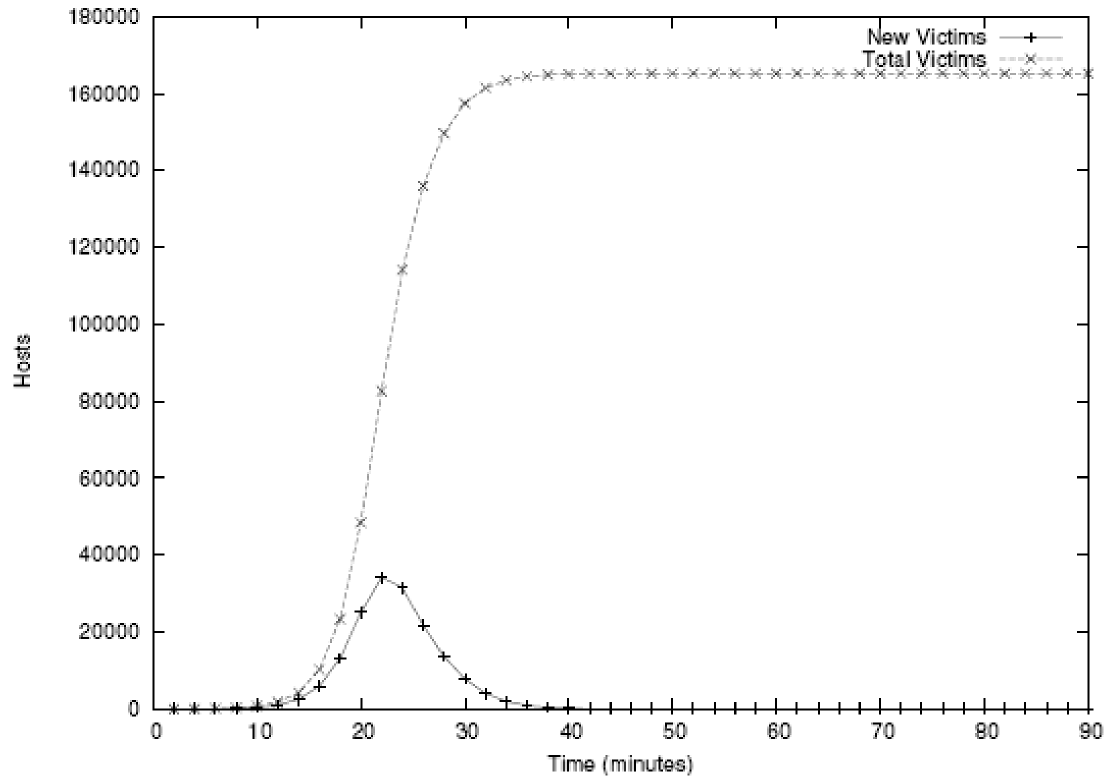
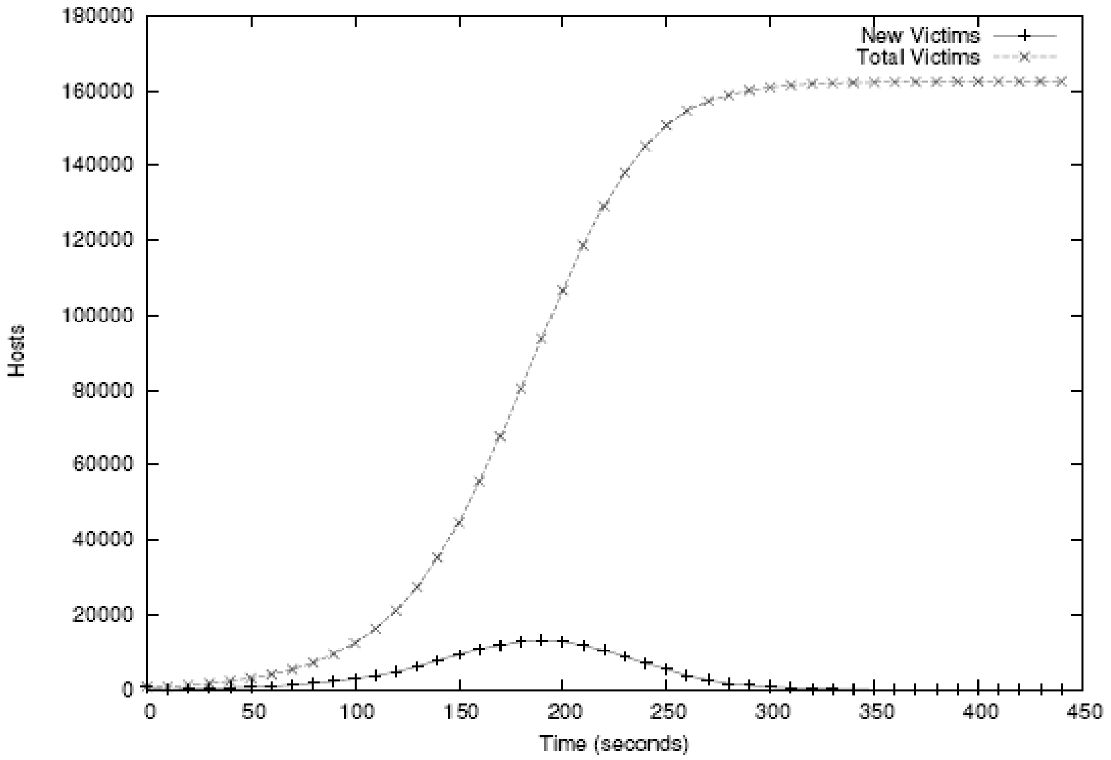


Figure 8: Local preference with multi-threading and short timeouts

Spreading - V



- prescan to find better subspaces to generate IP addresses
- and with a large number of susceptible nodes
- Infected nodes are remembered and neglected
- multithread

Local vs global

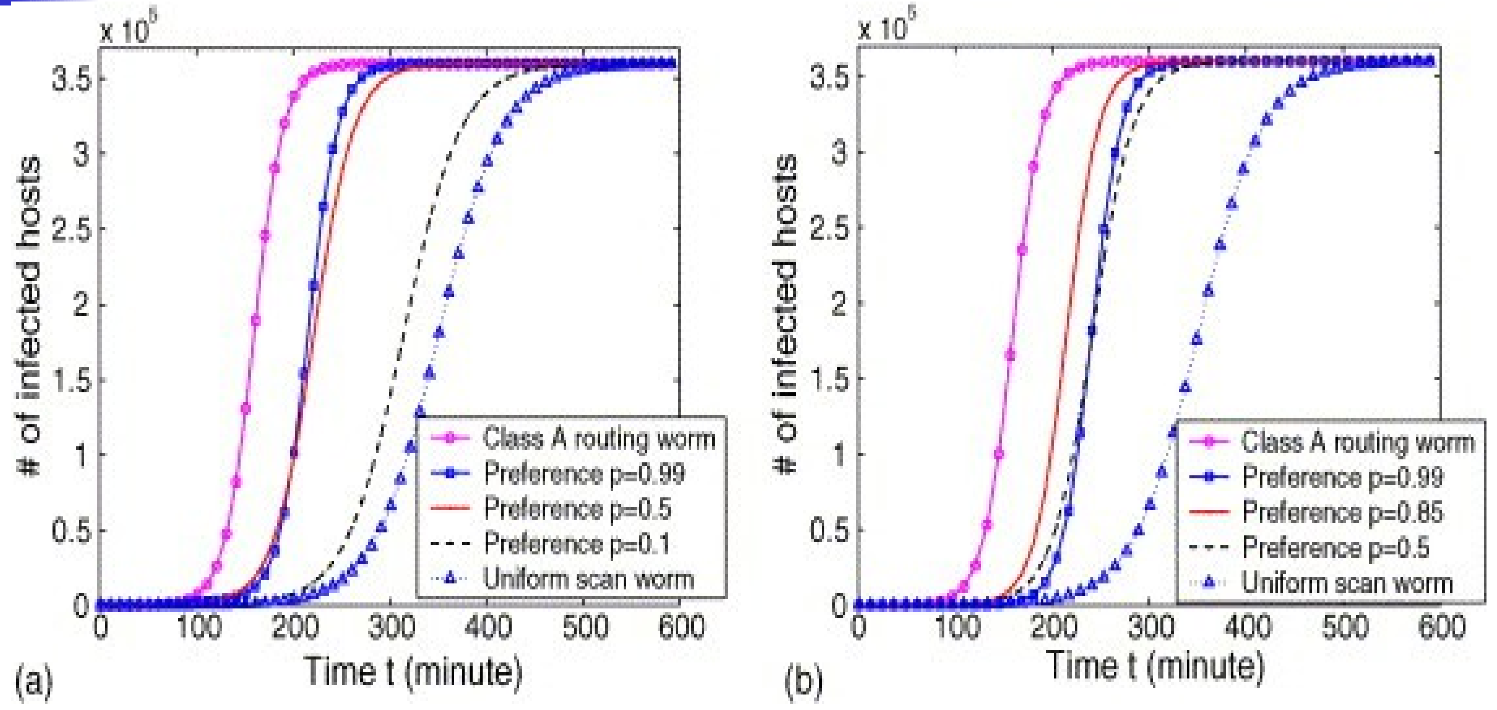
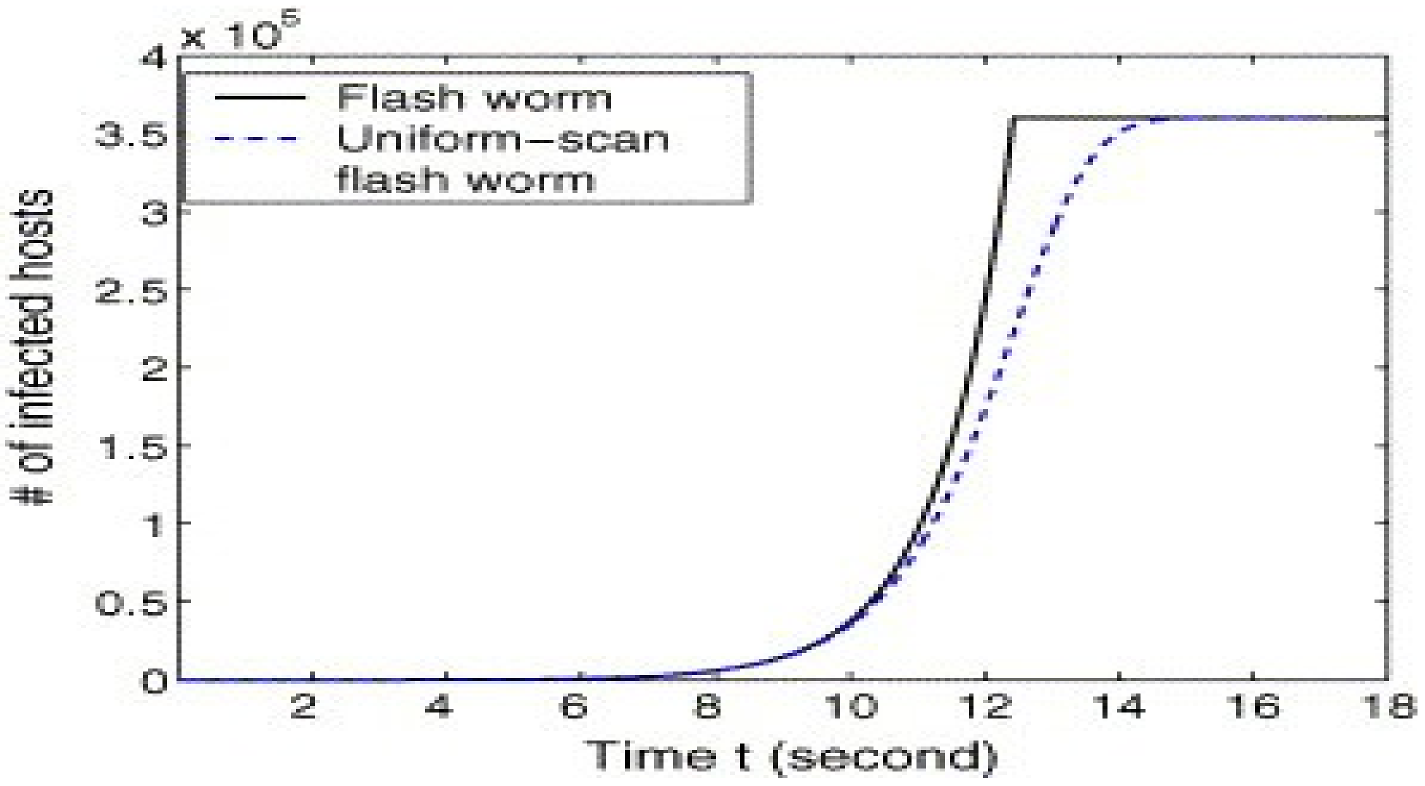


Fig. 5. Comparison of Code Red, a /8 routing worm, a local preference worm with different preference probabilities p.

(a) Local preference scan on "/8" network level (K=256, m=116).

(b) Local preference scan on "/16" network level (K=65,536, m=29,696).

Extreme optimization



The time scale has changed

Which address space?

- Some worms consider IP addresses
  - ⇒Any node can infect any other nodes
  - ⇒The addresses that are generated depend upon the adopted function and not upon the interconnection
  - ⇒Highly effective but high error rate
- Some worms consider logical addresses, ie the email addresses
  - ⇒A node can infect only nodes it already knowns
  - ⇒The interconnection structure that has to be considered is the logical one

Trojan horse

- A program that has a different goal from the expected one
- Its main goal is to implement a backdoor to enable illegal accesses to the system
- Governmental to acquire information and defeat encryption
- Malware

Trojan horse defence (wikipedia)

- This defense (SODDI, some other dude did it) typically involves defendant denial of responsibility for
  - (i) the presence of cyber contraband on the defendant's computer system;
  - (ii) commission of a cybercrime via the defendant's computer, on the basis that a malware or on some other perpetrator using such malware, was responsible for the offence in question.
- A modified use of the defense involves a defendant charged with a non-cyber crime admitting that whilst technically speaking the defendant may be responsible for the commission of the offence, he or she lacked the necessary criminal intent or knowledge on account of malware involvement.

“Reflections on Trusting Trust”

Ken Thompson’s 1983 Turing Award lecture

1. Added a backdoor-opening Trojan to login program
2. Anyone looking at source code would see this, so changed the compiler to add backdoor at compile-time
3. Anyone looking at compiler source code would see this, so changed the compiler to recognize when it’s compiling a new compiler and to insert Trojan into it

“The moral is obvious. You can’t trust code you did not totally create yourself. (Especially code from companies that employ people like me).”

Viruses

Virus propagates by infecting other programs

- Automatically creates copies of itself, but to propagate, a human has to run an infected program
- Self-propagating viruses are often called <u>worms</u>

Many propagation methods

- Insert a copy into every executable (.COM, .EXE)
- Insert a copy into boot sectors of disks
    - PC era: “Stoned” virus infected PCs booted from infected floppies, stayed in memory, infected every inserted floppy
- Infect common OS routines, stay in memory

First Virus: Creeper

Written in 1971 at BBN

Infected DEC PDP-10
  machines running TENEX OS

Jumped from machine to machine over ARPANET

- Copied its state over, tried to delete old copy

Payload: displayed a message
  “I’m the creeper, catch me if you can!”

Later, Reaper was written to hunt down Creeper

Polymorphic Viruses

Encrypted viruses: constant decryptor followed by the encrypted virus body

Polymorphic viruses: each copy creates a new random encryption of the same virus body

- Decryptor code constant and can be detected
- Historical note: "Crypto" virus decrypted its body by brute-force key search to avoid explicit decryptor code

Virus Detection

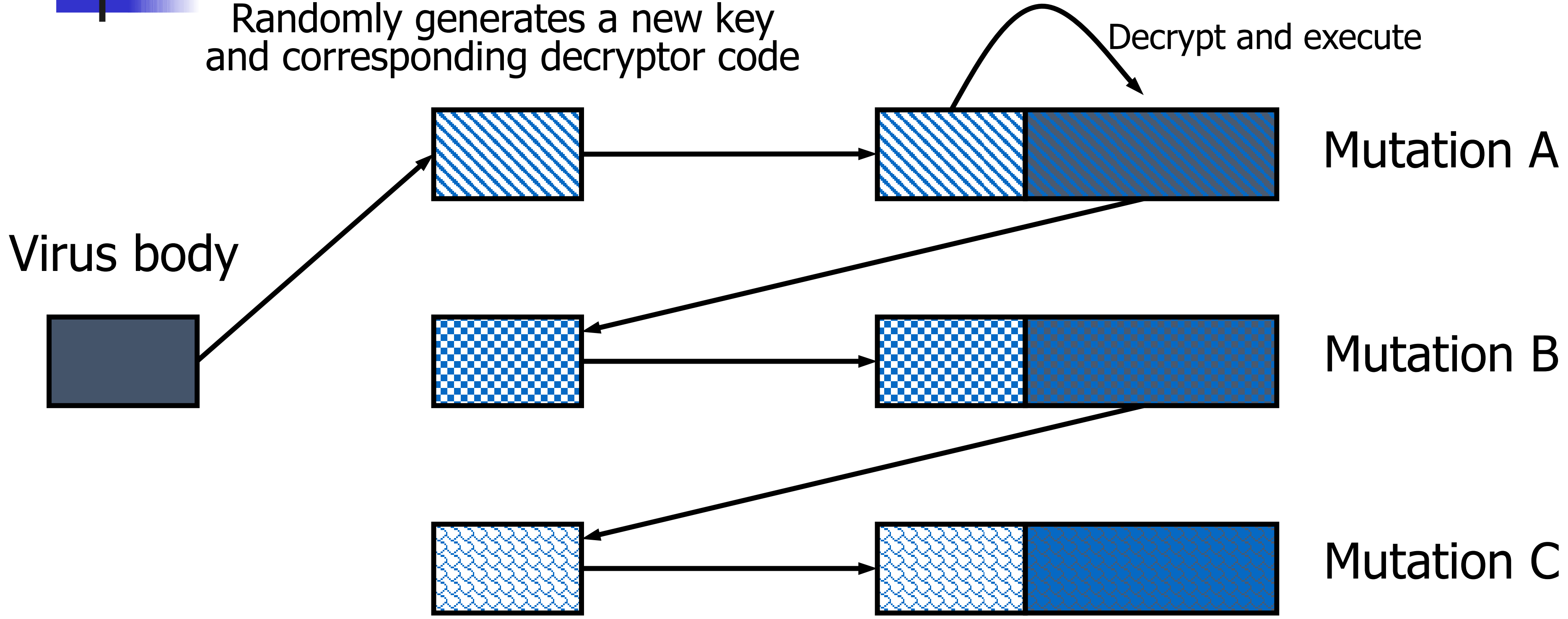
Simple anti-virus scanners

- Look for signatures (fragments of known virus code)
- Heuristics for recognizing code associated with viruses
  - Example: polymorphic viruses often use decryption loops
- Integrity checking to detect file modifications
  - Keep track of file sizes, checksums, keyed HMACs of contents

Generic decryption and emulation

- Emulate CPU execution for a few hundred instructions, recognize known virus body after it has been decrypted
- Does not work very well against viruses with mutating bodies and viruses not located near beginning of infected executable

Virus Detection by Emulation


Randomly generates a new key and corresponding decryptor code
Decrypt and execute
Virus body
Mutation A
Mutation B
Mutation C

To detect an unknown mutation ▨ of a known virus ■ , emulate CPU execution of ▨ until the current sequence of instruction opcodes matches the known sequence for virus body ■

Metamorphic Viruses

Obvious next step: mutate the virus body, too

Apparition: an early Win32 metamorphic virus

- Carries its source code (contains useless junk)
- Looks for compiler on infected machine
- Changes junk in its source and recompiles itself
- New binary copy looks different!

Mutation is common in macro and script viruses

- A macro is an executable program embedded in a word processing document (MS Word) or spreadsheet (Excel)
- Macros and scripts are usually interpreted, not compiled

Obfuscation and Anti-Debugging

Common in all kinds of malware

Goal: prevent code analysis and signature-based detection, foil reverse-engineering

Code obfuscation and mutation

- Packed binaries, hard-to-analyze code structures
- Different code in each copy of the virus
    - Effect of code execution is the same, but this is difficult to detect by passive/static analysis (undecidable problem)

Detect debuggers and virtual machines, terminate execution

Mutation Techniques

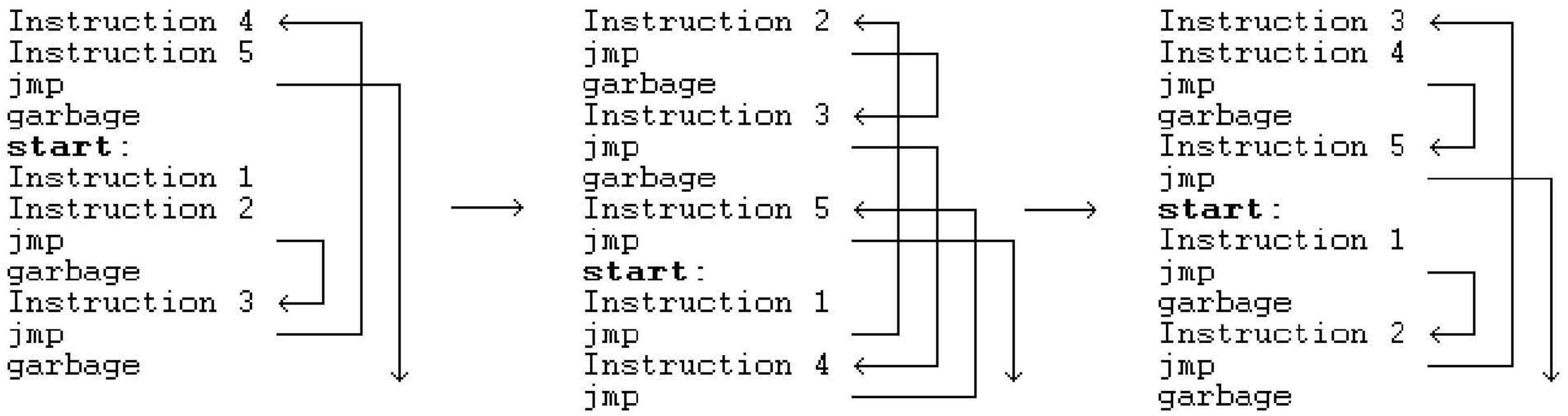
Real Permutating Engine/RPME, ADMutate, etc.

Large arsenal of obfuscation techniques

- Instructions reordered, branch conditions reversed, different register names, different subroutine order
- Jumps and NOPs inserted in random places
- Garbage opcodes inserted in unreachable code areas
- Instruction sequences replaced with other instructions that have the same effect, but different opcodes
  - Mutate SUB EAX, EAX into XOR EAX, EAX or
    MOV EBP, ESP into PUSH ESP; POP EBP

There is no constant, recognizable virus body

Example of Zperm Mutation



From Szor and Ferrie, “Hunting for Metamorphic”

Putting It All Together: Zmist

Designed in 2001 by the Russian virus writer Z0mbie of "Total Zombification" fame

Technique: code integration

- Virus merges itself into the instruction flow of its host
- "Islands" of code are integrated into random locations in the host program and linked by jumps
- When/if virus code is run, it infects every available portable executable
  - A randomly inserted virus entry point may not be reached in a particular execu

Putting It All Together: Zmist

Designed in 2001 by the Russian virus writer Z0mbie of “Total Zombification” fame

Technique: code integration

- Virus merges itself into the instruction flow of its host
- “Islands” of code are integrated into random locations in the host program and linked by jumps
- When/if virus code is run, it infects every available portable executable
  - A randomly inserted virus entry point may not be reached in a particular execution

MISTFALL Disassembly Engine

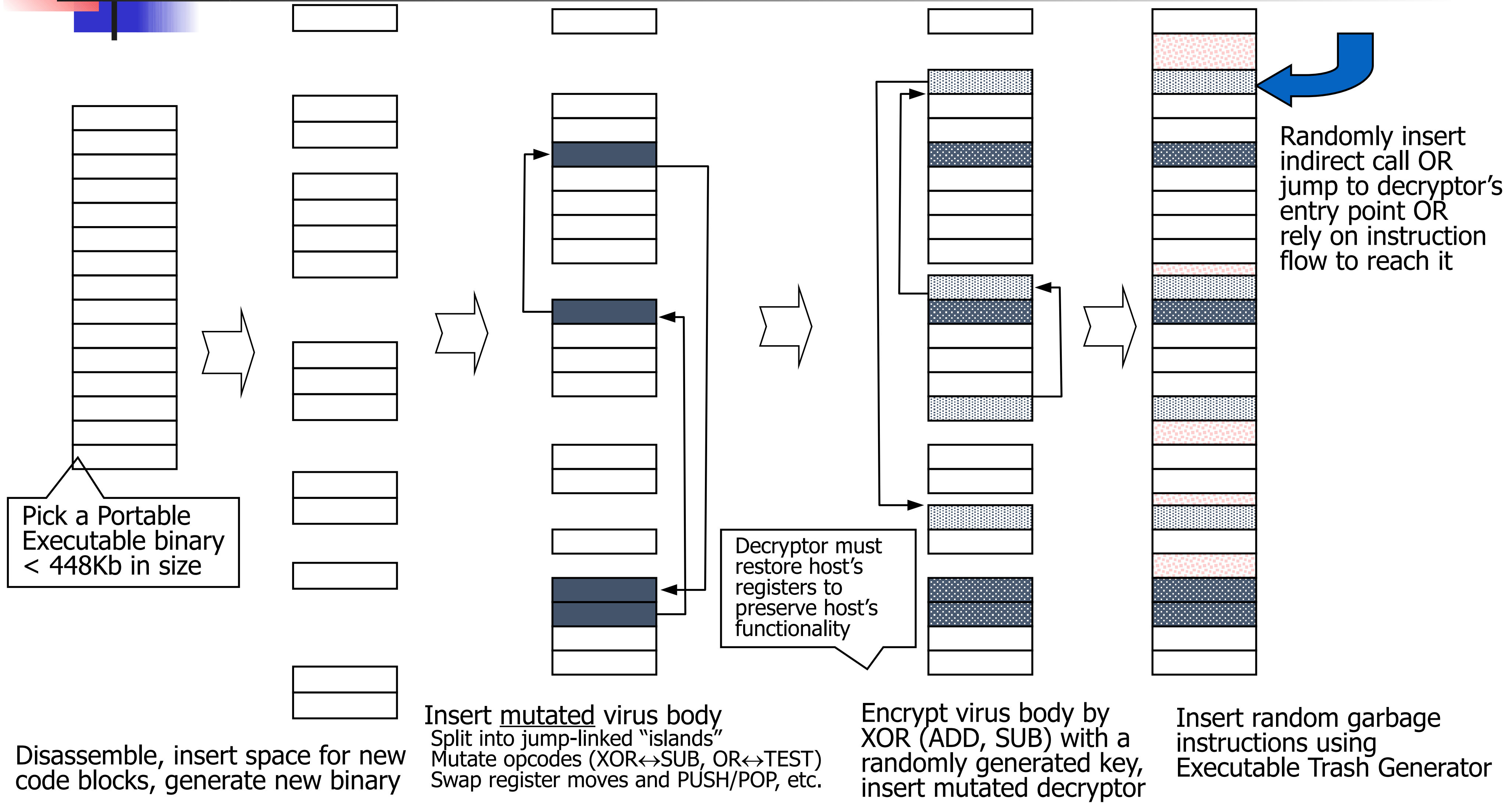
To integrate itself into host's instruction flow, virus must disassemble and rebuild host binary

Tricky - addresses are based on offsets, must be recomputed when new instructions are inserted

Iterative process: rebuild with new addresses, see if branch destinations changed, rebuild again

- Requires 32MB of RAM and explicit section names (DATA, CODE, etc.) in the host binary – doesn't work with every file

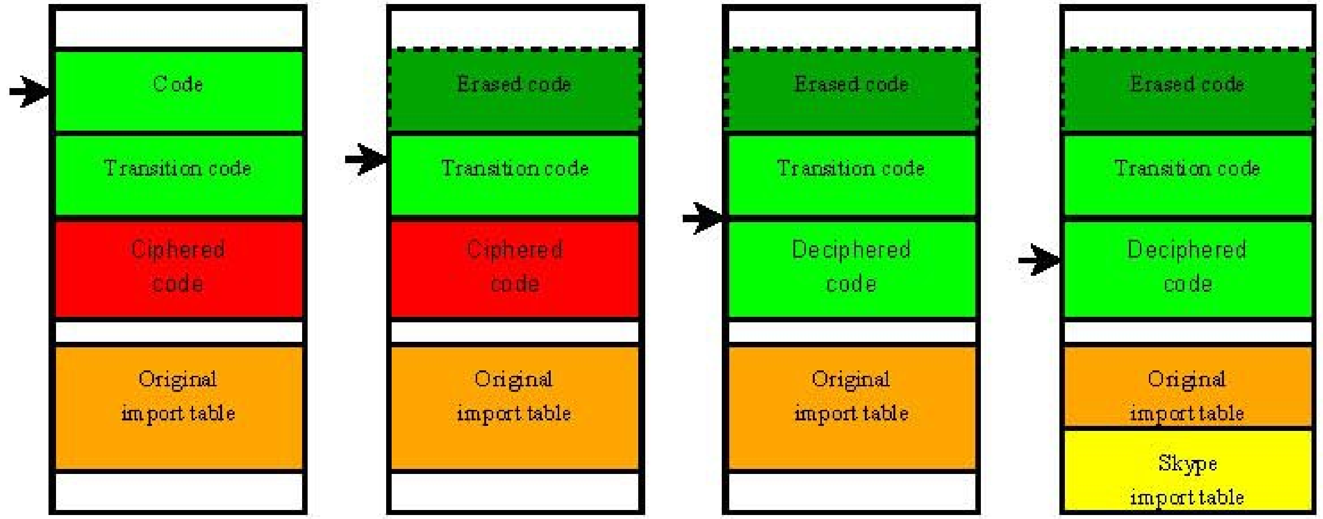
Simplified Zmist Infection Process


Pick a Portable Executable binary < 448Kb in size
Disassemble, insert space for new code blocks, generate new binary
Insert mutated virus body
Split into jump-linked "islands"
Mutate opcodes (XOR↔SUB, OR↔TEST)
Swap register moves and PUSH/POP, etc.
Decryptor must restore host's registers to preserve host's functionality
Encrypt virus body by XOR (ADD, SUB) with a randomly generated key, insert mutated decryptor
Randomly insert indirect call OR jump to decryptor's entry point OR rely on instruction flow to reach it
Insert random garbage instructions using Executable Trash Generator

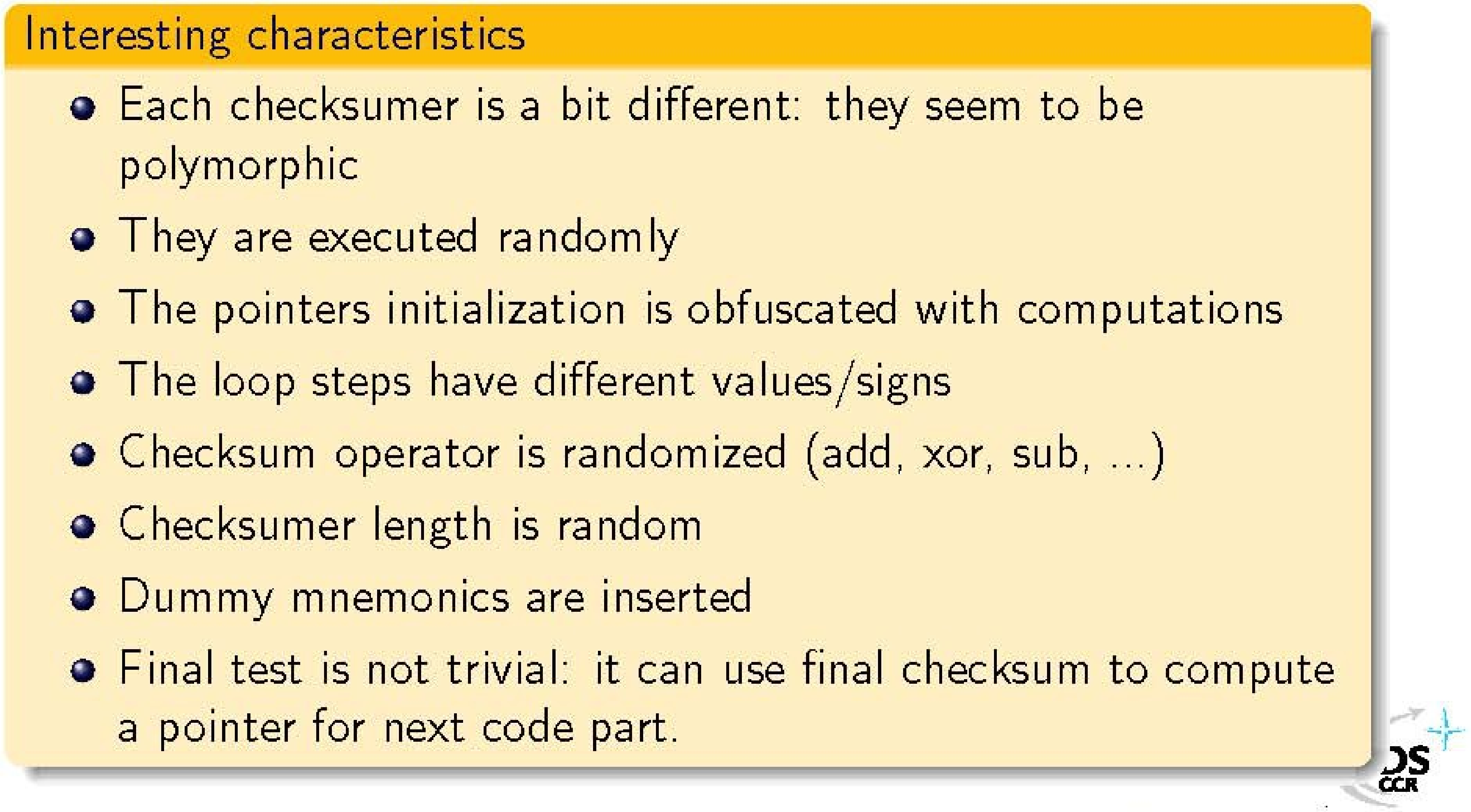
Legal obfuscation : Skype

Anti-dumping tricks

1. The program erases the beginning of the code
2. The program deciphers encrypted areas
3. Skype import table is loaded, erasing part of the original import table



Skype: Code Integrity Checking



Interesting characteristics

- Each checksumer is a bit different: they seem to be polymorphic
- They are executed randomly
- The pointers initialization is obfuscated with computations
- The loop steps have different values/signs
- Checksum operator is randomized (add, xor, sub, ...)
- Checksumer length is random
- Dummy mnemonics are inserted
- Final test is not trivial: it can use final checksum to compute a pointer for next code part.

Skype: Anti-Debugging

Counter measures

- When it detects an attack, it traps the debugger :
  - registers are randomized
  - a random page is jumped into
- It's is difficult to trace back the detection because there is no more stack frame, no EIP, ...

```
pushf
pusha
mov       save_esp , esp
mov       esp , ad_alloc?
add       esp , random_value
sub       esp , 20h
popa
jmp       random_mapped_page
```

Skype: Control Flow Obfuscation (1)

Code indirection calls

```
mov     eax, 9FFB40h
sub     eax, 7F80h
mov     edx, 7799C1Fh
mov     ecx, [ebp-14h]
call    eax ; sub_9F7BC0
neg     eax
add     eax, 19C87A36h
mov     edx, 0CCDACEF0h
mov     ecx, [ebp-14h]
call    eax
; eax = 009F8F70
```

```
sub_9F8F70:
mov     eax, [ecx+34h]
push    esi
mov     esi, [ecx+44h]
sub     eax, 292C1156h
add     esi, eax
mov     eax, 371509EBh
sub     eax, edx
mov     [ecx+44h], esi
xor     eax, 40F0FC15h
pop     esi
retn
```

Principle

Each call is dynamically computed: difficult to follow statically

Determined conditional jumps

```
...
if ( sin(a) == 42 ) {
        do_dummy_stuff();
}
go_on();
...
```

Skype: Control Flow Obfuscation (2)

Execution flow rerouting

```
lea
edx ,  [esp+4+var_4]
add        eax ,  3D4D101h
push       offset  area
push       edx
mov
[esp+0Ch+var_4] ,  eax
call       RaiseException
rol        eax ,  17h
xor        eax ,  350CA27h
pop        ecx
```

- Sometimes, the code raises an exception
- An error handler is called
- If it's a fake error, the handler tweaks memory addresses and registers

$\Longrightarrow$ back to the calling code

Rootkits

Rootkit is a set of trojan system binaries

Main characteristic: <u>stealthiness</u>

- Create a hidden directory
  - /dev/.lib, /usr/src/.poop and similar
  - Often use invisible characters in directory name (why?)
- Install hacked binaries for system programs such as netstat, ps, ls, du, login
-
-
- Modified binaries have same checksum as originals
  - What should be used instead of checksum?

Can't detect attacker's processes, files or network connections by running standard UNIX commands!

Function Hooking

Rootkit may “re-route” a legitimate system function to the address of malicious code

Pointer hooking

- Modify the pointer in OS’s Global Offset Table, where function addresses are stored

“Detour” or “inline” hooking

- Insert a jump in first few bytes of a legitimate function
- This requires subverting memory protection

Modifications may be detectable by a clever rootkit detector

Kernel Rootkits

Get loaded into OS kernel as an external module

- For example, via compromised device driver or a badly implemented “digital rights” module (e.g., Sony XCP)

Replace addresses in system call table, interrupt descriptor table, etc.

If kernel modules disabled, directly patch kernel memory through /dev/kmem (SucKIT rootkit)

Inject malicious code into a running process via PTRACE_ATTACH and PTRACE_DETACH

- Security and antivirus software are often the first injection targets

Mebroot (Windows)

Replaces the host’s Master Boot Record (MBR)

- First physical sector of the hard drive
- Launches before Windows loads

No registry changes, very little hooking

Stores data in physical sectors, not files

- Invisible through the normal OS interface

Uses its own version of network driver API to send and receive packets

- Invisible to “personal firewall” in Windows

Used in Torpig botnet

Detecting Rootkit's Presence

Sad way to find out

- Run out of physical disk space because of sniffer logs
- Logs are invisible because du and ls have been hacked

Manual confirmation

- Reinstall clean ps and see what processes are running

Automatic detection

- Rootkit does not alter the data structures normally used by netstat, ps, ls, du, ifconfig
- Host-based intrusion detection can find rootkit files
  - …assuming an updated version of rootkit did not disable the intrusion detection system!

Remote Administration Tools

Legitimate tools are often abused

- Citrix MetaFrame, WinVNC, PC Anywhere
  - Complete remote control over the machine
  - Easily found by port scan (e.g., port 1494 – Citrix)
- Bad installations, crackable password authentication
  - “The Art of Intrusion” – hijacking remote admin tools to break into a cash transfer company, a bank’s IBM AS/400 server

Semi-legitimate tools

- Back Orifice, NetBus
- Rootkit-like behavior: hide themselves, log keystrokes
- Considered malicious by anti-virus software

Communicating Via Backdoors

All sorts of standard and non-standard tunnels

SSH daemons on a high port

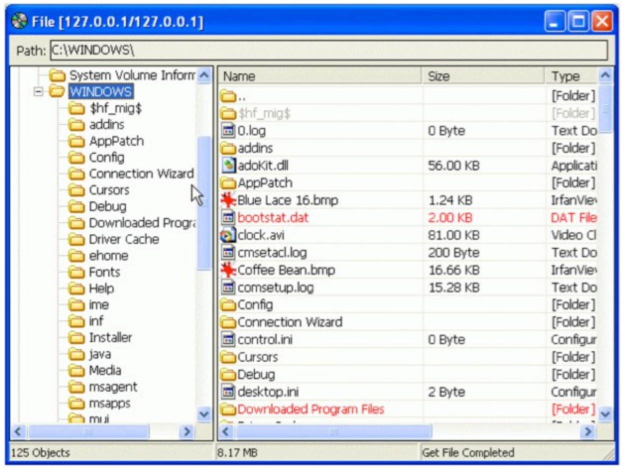
- Communication encrypted $\Rightarrow$ hard to recognize for a network-based intrusion detector
- Hide SSH activity from the host by patching netstat

UDP listeners

Passively sniffing the network for master's commands

RAT Capabilities

- "Dropper" program installs RAT DLL, launches it as persistent Windows service, deletes itself
- RAT notifies specified C&C server, waits for instructions
- Attacker at C&C server has full control of the infected machine, can view files, desktop, manipulate registry, launch command shell

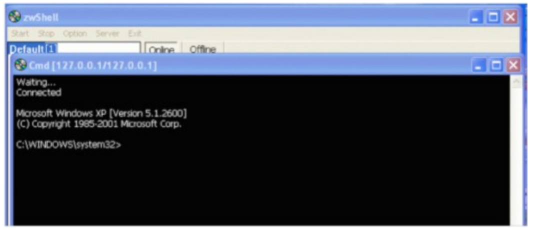

File [127.0.0.1/127.0.0.1]

Path: C:\WINDOWS\

System Volume Inform
WINDOWS
hf_mig
addins
AppPatch
Config
Connection Wizard
Cursors
Debug
Downloaded Progr
Driver Cache
ehome
Fonts
Help
ime
inf
Installer
java
Media
msagent
msapps
mui

Name	Size	Type
..		[Folder]
hf_mig		[Folder]
0.log	0 Byte	Text Do
addins		[Folder]
adoKit.dll	56.00 KB	Applicati
AppPatch		[Folder]
Blue Lace 16.bmp	1.24 KB	IrfanVie
bootstat.dat	2.00 KB	DAT File
clock.avi	81.00 KB	Video Cl
cmsetacl.log	200 Byte	Text Do
Coffee Bean.bmp	16.66 KB	IrfanVie
comsetup.log	15.28 KB	Text Do
Config		[Folder]
Connection Wizard		[Folder]
control.ini	0 Byte	Configur
Cursors		[Folder]
Debug		[Folder]
desktop.ini	2 Byte	Configur
Downloaded Program Files		[Folder]

125 Objects | 8.17 MB | Get File Completed



Cmd [127.0.0.1/127.0.0.1]

```
Waiting...
Connected

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>
```

Hybrid

- Most malware current integrates all the previous behavior
- Software with an opportunistic approach to spread to other nodes
  - Usb
  - Share
  - Mail
  - Attack
  - ....

Autonomous Hybrid

- They can transmit themselves to other nodes without exploiting the node resources
- Even if the node does not exchange email, it can
  - Trasmit email from the node
  - Hide in the mail