

Attack Graphs

Identifying Critical Vulnerabilities Within An Organization

10 March 2004

Todd Heberlein

Melissa Danforth

Senthil

Ebrima

Tye Stallard

Primary Goal

Maximizing security of a network
using limited resources.

$$\text{Cost(attack)} \gg \text{Cost(patch)}$$

Preventing attacks is better than
detecting successful attacks.

Vulnerability Scanners

- Nessus
- ISS Internet Scanner
- Cisco Security Scanner
- Saint
- Older versions
 - COPS
 - SATAN

So What?

- Point-based solutions. They identify individual variables with no understanding of the relationship between variables.
- No understanding of how any particular variable may or may not affect the organization's mission
- Overwhelming amounts of data. Where does a security administrator start?

Example Nessus Scan

Elapsed Time : 00:48:07

Total security holes found : 255

high severity : 40

low severity : 117

informational : 98

So What?

- Can any of these holes reveal state-protected privacy information?
- Can any of these holes affect faculty candidate vote database?
- Can any of these holes allow the CS web page to be vandalized?
- How can low severity and information vulnerabilities affect these the network security?

Prioritizing Fixes

- Identify the vulnerabilities most likely to be exploited and focus on fixing those first.
 - Focus of our TrendCenter approach
 - SANS Top-n based on our approach
- Identify vulnerabilities that put mission at risk.
- Identify vulnerabilities that provide the greatest access to the rest of the network.

What Is An Attack Graph?

An attack graph is a set of actions that increase an adversaries capabilities.

The graph can focus on whether a certain set of initial capabilities can eventually lead to some critical capability.

The graph can focus on the extent to which an adversary can penetrate a network given an initial set of capabilities.

Traditional graphed-based analyses can be applied to identify optimal changes to the network.

Approaches to Building Graphs

How To Build Graphs

- Manual examination
- Roll your own code
- Use expert system shells (Jess, Clips)
- Use Symbolic Model Verifiers (SMVs)
- Use theorem provers

Manual Approach

- Criticisms
 - Few experts available
 - Red Teams can be expensive
 - Tedious
 - Error-prone
 - Impractical for large networks
 - No formal claims

Roll Your Own Code Approach

- Criticisms
 - Requires significant effort
 - duplicates efforts of SMVs
 - Error prone
 - library argument
 - No formal claims

Symbolic Model Verifier Approach

- Benefits
 - Builds on existing engines
 - Recent advances support scaling to large state spaces (possible infinite spaces)
- Formal claims can be made
 - Results are exhaustive - all possible attack paths are identified
 - Results are succinct - only states reachable from the target are identified

Requires & Provides

Kuang Example

Replace /d	Requires
Replace /d/f	Provides

Replace /etc/passwd	Requires
Become root	Provides

GMU/NuSMV Example

attack sshd-buffer-overflow is

intruder preconditions

$plvA(S) \geq \text{user}$

$plvA(T) < \text{root}$

network preconditions

sshT

$R(S, T, sp)$

intruder effects

$plvA(T) := \text{root}$

network effects

-sshT

end

Requires

Provides

JESS/Clips Example

```
(defrule sshd-buffer-overflow
```

```
  (priv ?s ?v)  
  (test (>= ?v 1))  
  ?plvl <- (priv ?t ?p)  
  (test (< ?p 2))  
  ?sshd <- (var ?t ssh)  
  (connect ?s ?t ssh)  
  (not (ids ?s ?t ssh))
```

```
==>
```

```
  (retract ?plvl)  
  (retract ?sshd)  
  (addert (priv ?t 2))
```

```
)
```

Requires

Provides

JIGSAW Example

```
concept RSH_Connection_Spoofing is

  requires
    Trusted_Partner:    TP;
    Service_Active:     SA;
    PreventPacketSend:  PPS;
    extern SeqNumProbe: SNP;
    ForgedPacketSend:   FPS;
  with
    TP.service is RSH,           #- The service in the trust relation is RSH
    PPS.host is TP.trusted,      #- The blocked host is the trusted partner
    FPS.dst.host is TP.trustor,  #- The spoofed packets are sent to the trustor
    SNP.dst.host is TP.trustor,  #- The probed host is the trustor
    FPS.src is [ND.host,PPS.port] #- claimed source of forged packets is blocked

    SNP.dst is [SA.host,SA.port] #- The probed host must be running RSH on the
    SA.port is TCP\RSH,          #- normal port
    SA.service is RSH,          #-

    SNP.dst is FPS.dest         #- probed host must be where forged packets are sent

    active(FPS) during active(PPS) #- forged packets must be sent while DOS is active
  end;

  provides
    push_channel:    PSC;
    remote_execution: REX;
  with
    PSC.from <- FPS.true_src;  #- Capability to move code from attacker to RSH server
    PSC.to   <- FPS.dst;       #-
    PSC.using <- RSH;         #-

    REX.from <- FPS.true_src;  #- Capability to execute code on remote host
    REX.to   <- FPS.dst;       #-
    REX.using <- RSH;         #-
  end;

  action
    true -> report ("RSH Connection Spoofing: TP.hostname")
  end;
end.
```

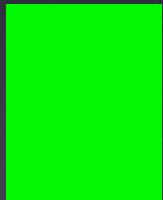
Requires

Provides

Common Architecture

Applies rules to fact base to
change fact base

Engine



Fact Base

Describes state
of network



Transition Rules

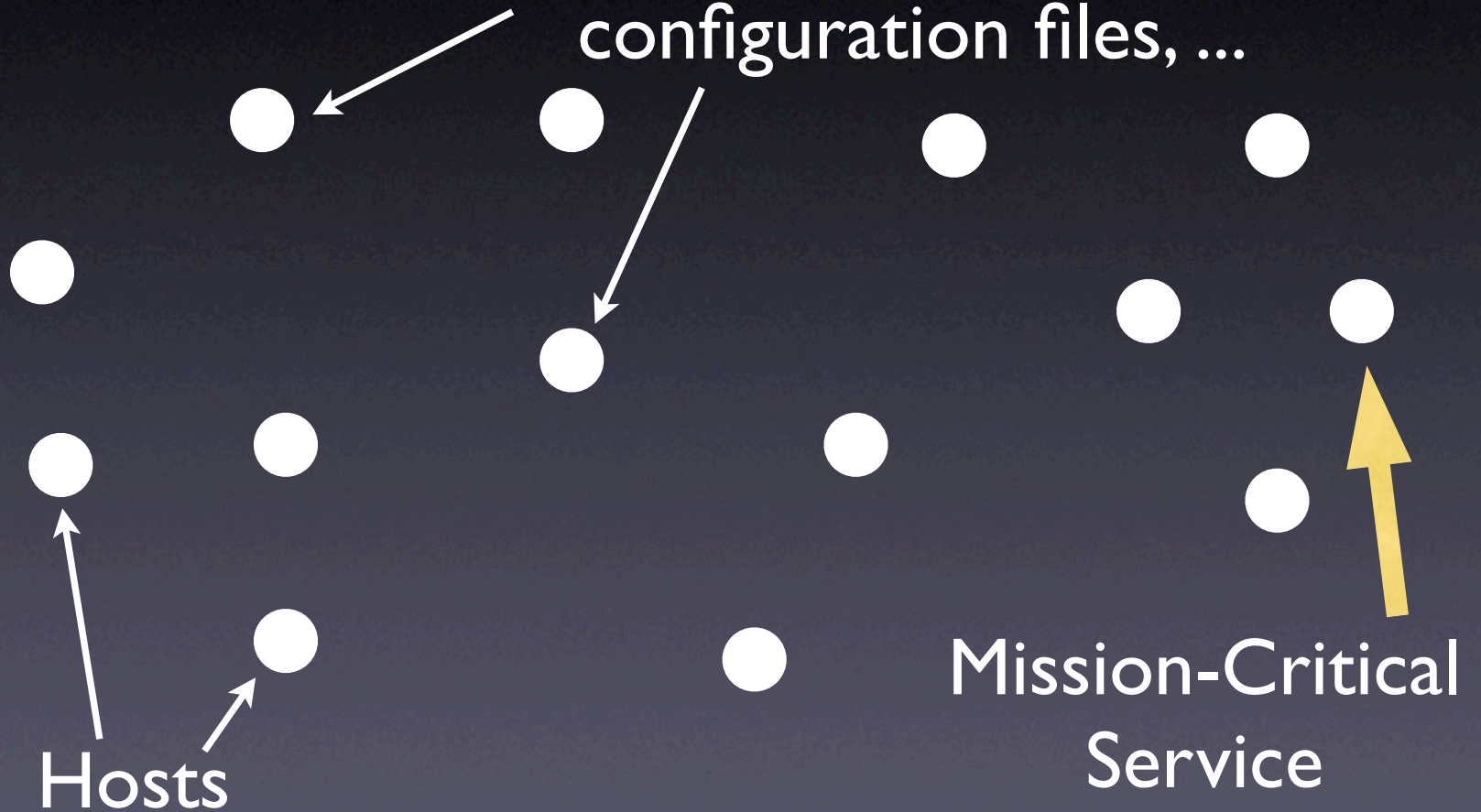
Describes
exploits



Forward vs Backward Chaining

Network Model

Or, could be processes, program,
configuration files, ...



Forward Chaining

Stage 0



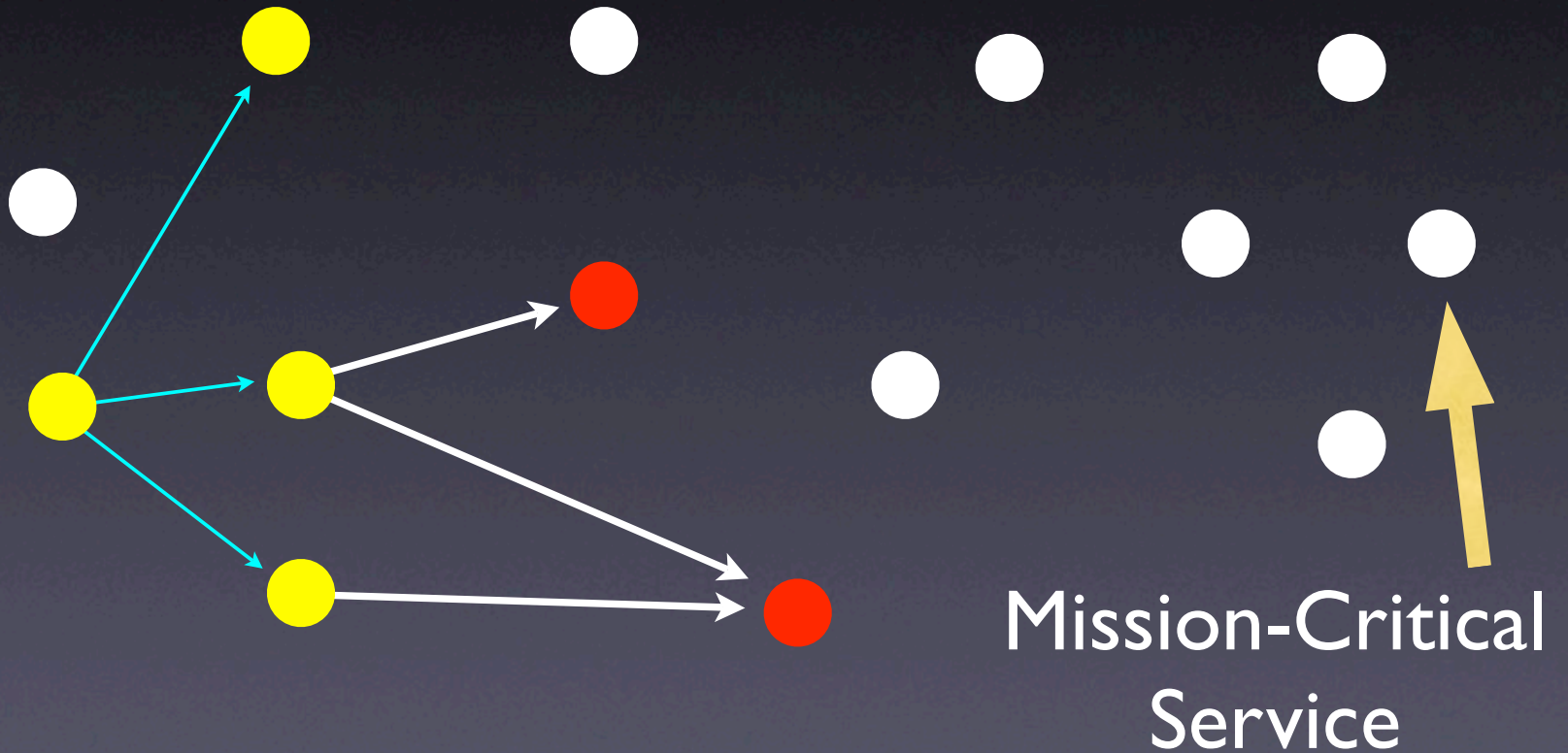
Forward Chaining

Stage I



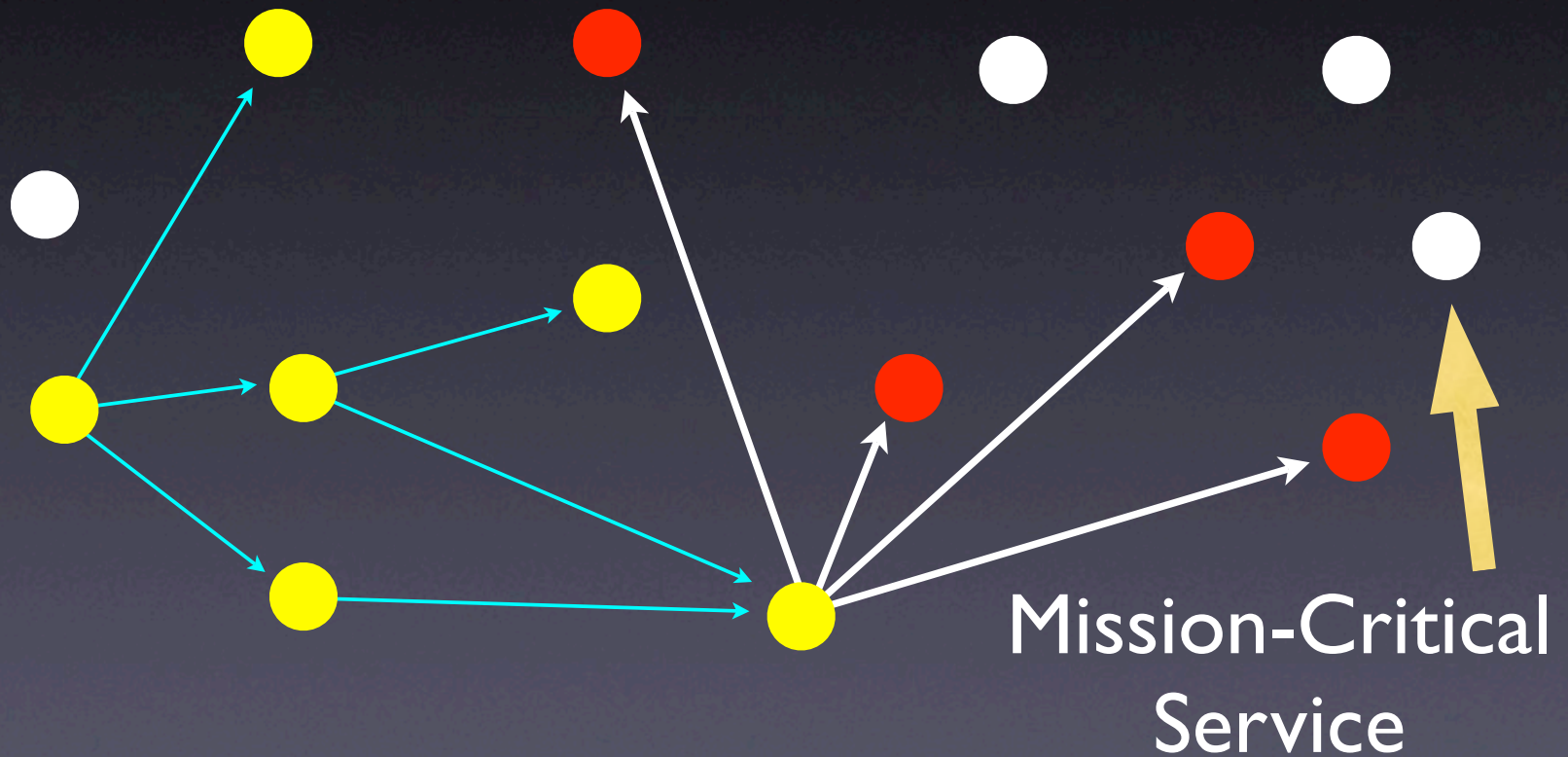
Forward Chaining

Stage 2



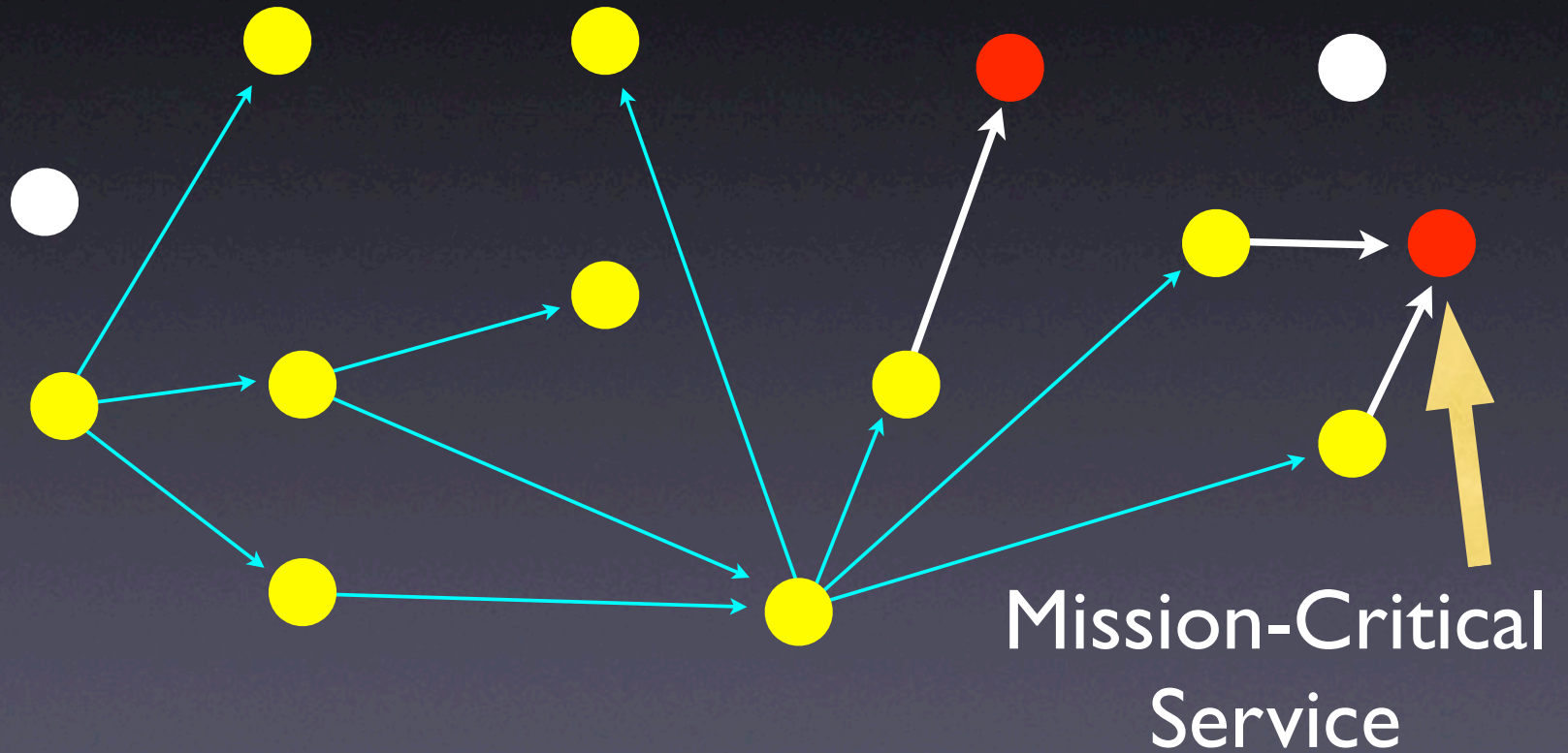
Forward Chaining

Stage 3



Forward Chaining

Stage 4



Backward Chaining

Stage 0



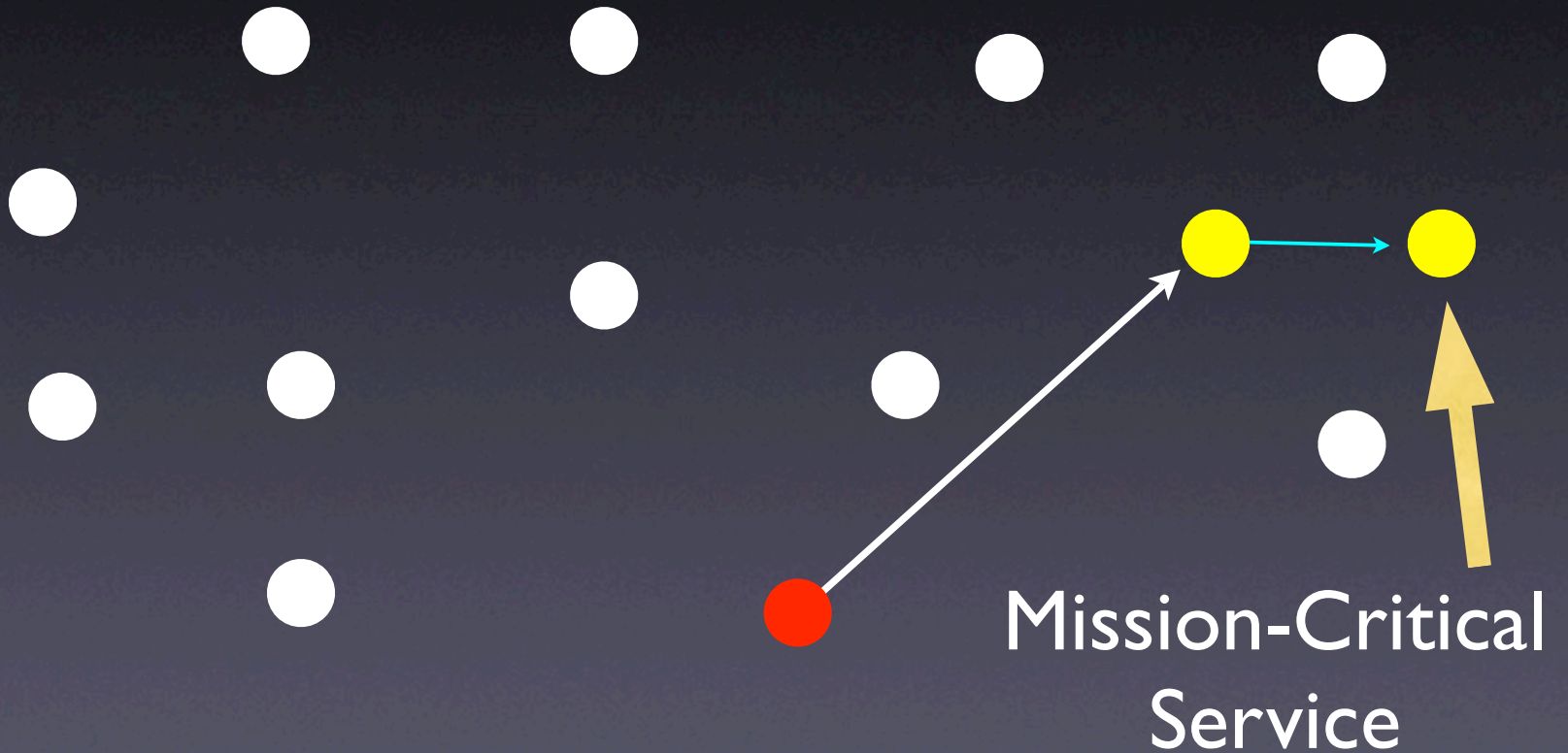
Backward Chaining

Stage I



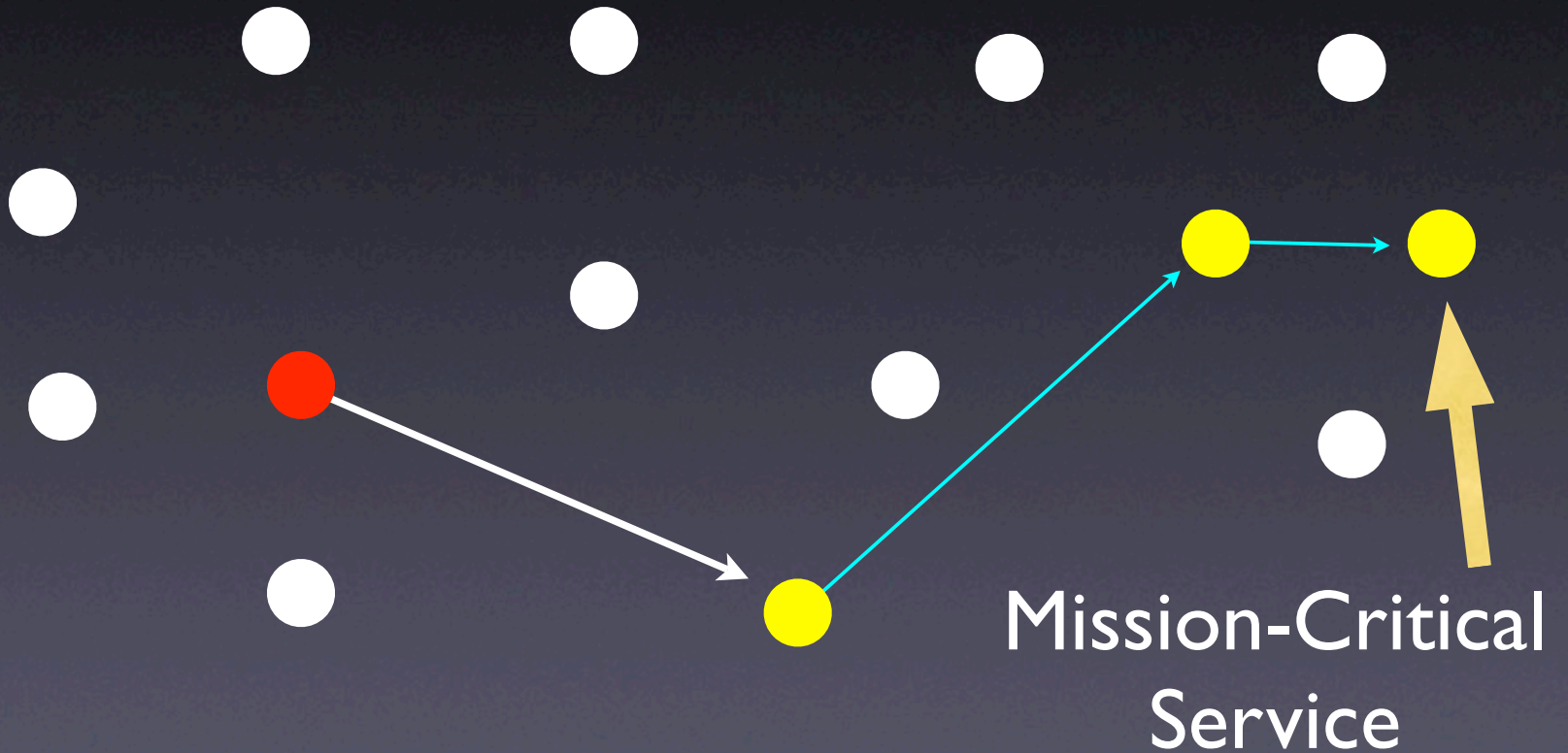
Backward Chaining

Stage 2



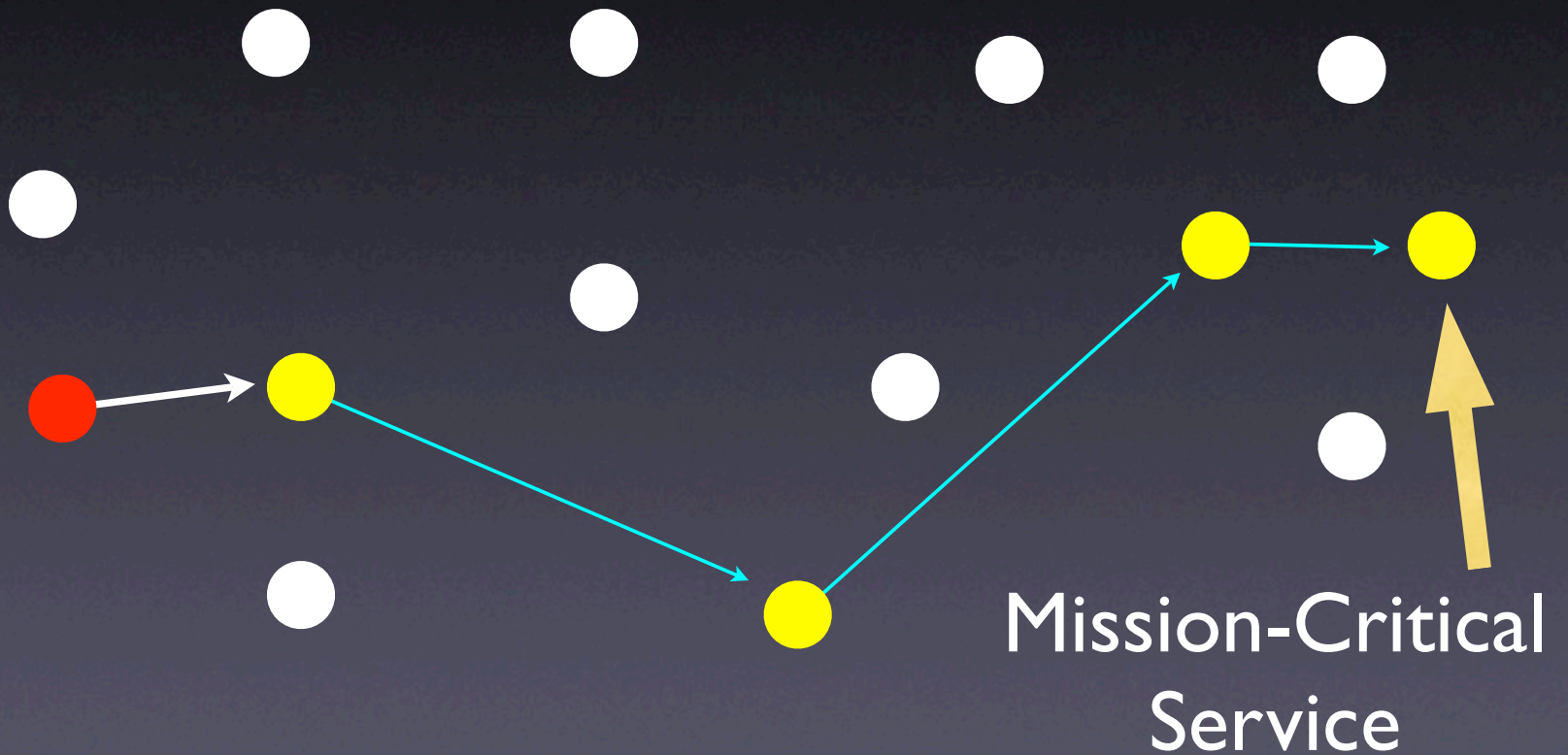
Backward Chaining

Stage 3



Backward Chaining

Stage 4



CMU/LL Approach

Model-based Approach

- Used NuSMV model checker
- Describe state with XML and then translates to SMV.
 - XML file has 239 lines (4 pages)
 - SMV file has 861 lines (14 pages)
- Supports non-monotonic attacks
- All (reachable) states are tracked

GMU/NuSMV Example

attack sshd-buffer-overflow is

intruder preconditions

$plvA(S) \geq \text{user}$

$plvA(T) < \text{root}$

network preconditions

sshT

$R(S, T, sp)$

intruder effects

$plvA(T) := \text{root}$

network effects

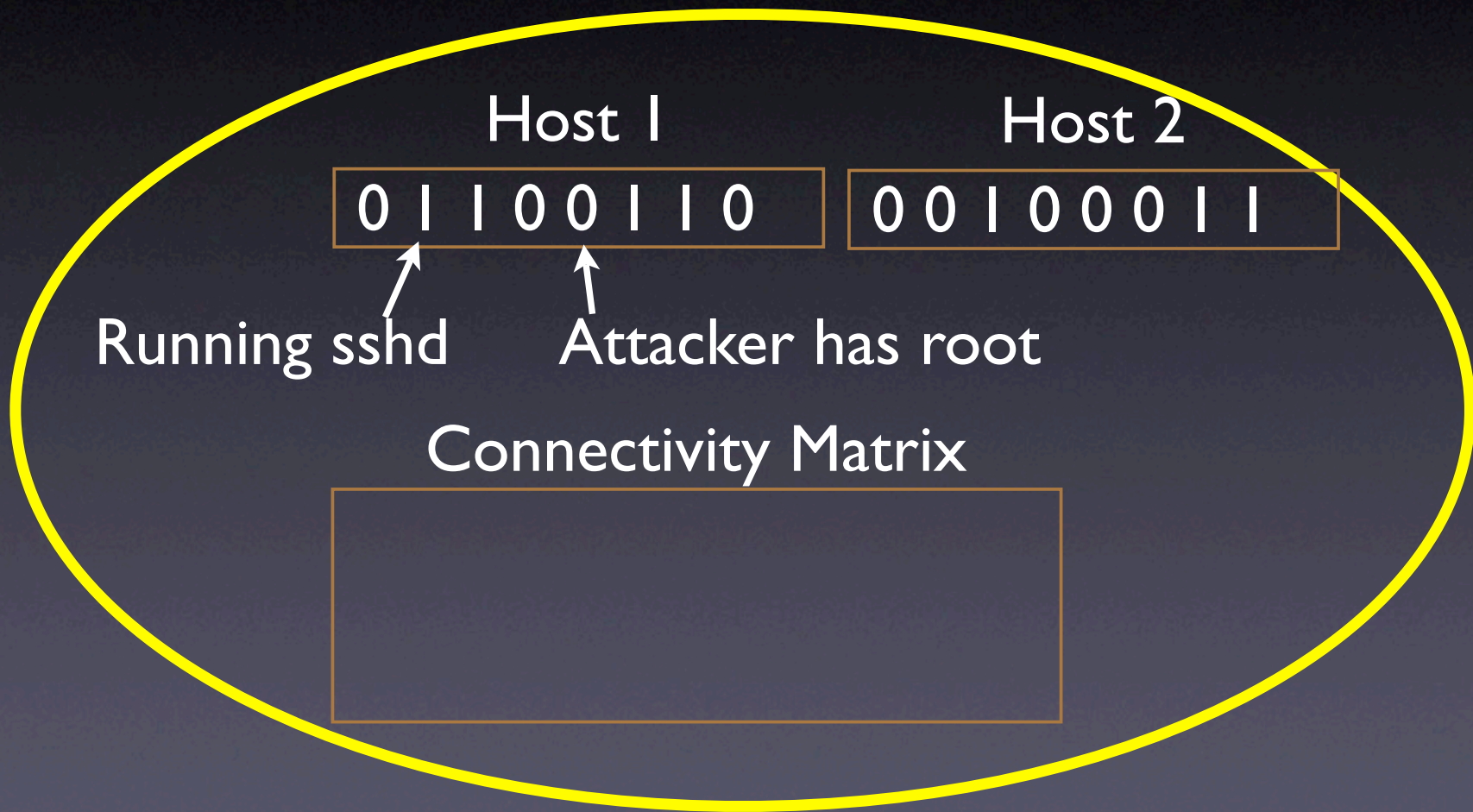
-sshT

end

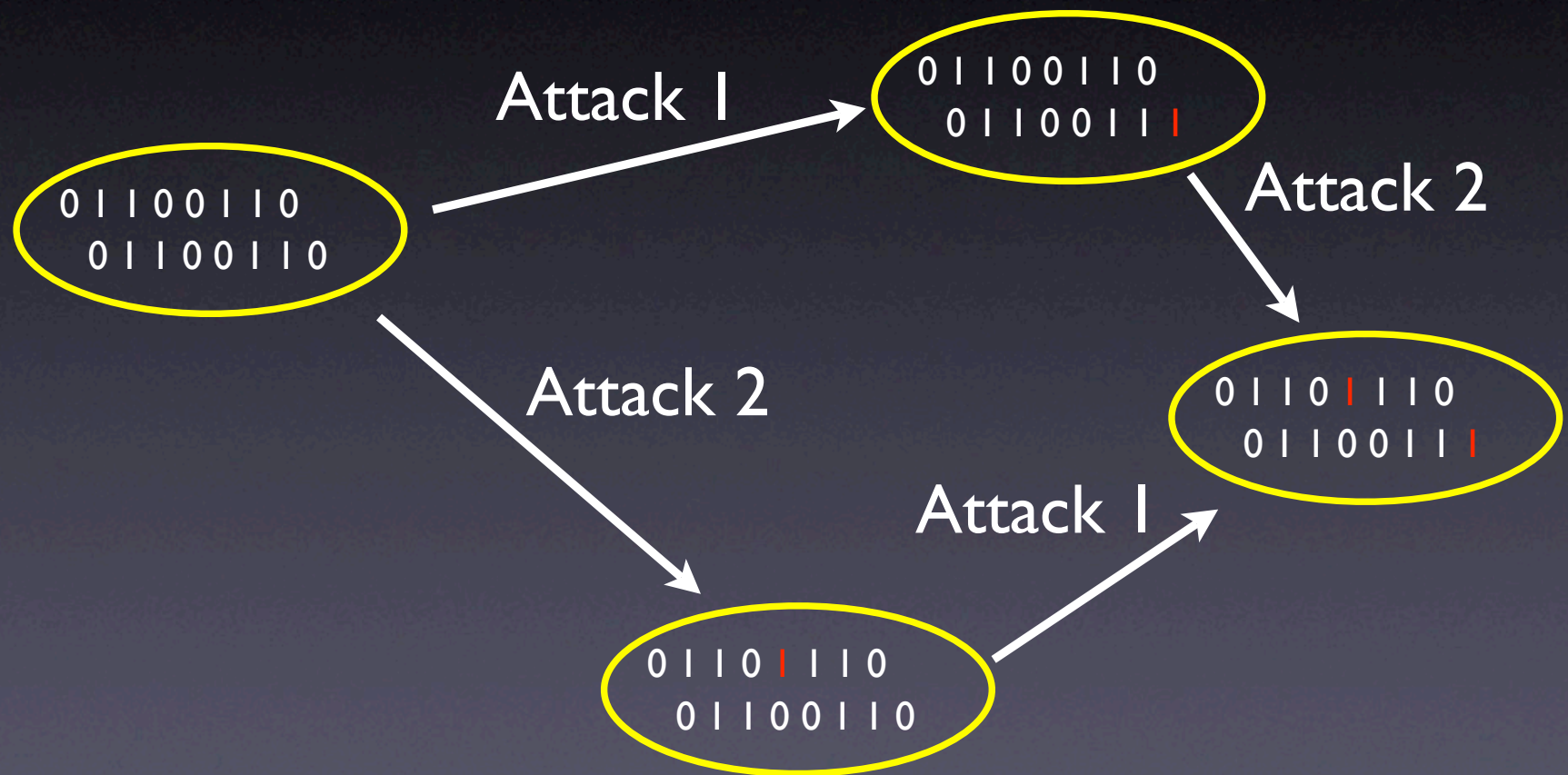
Requires

Provides

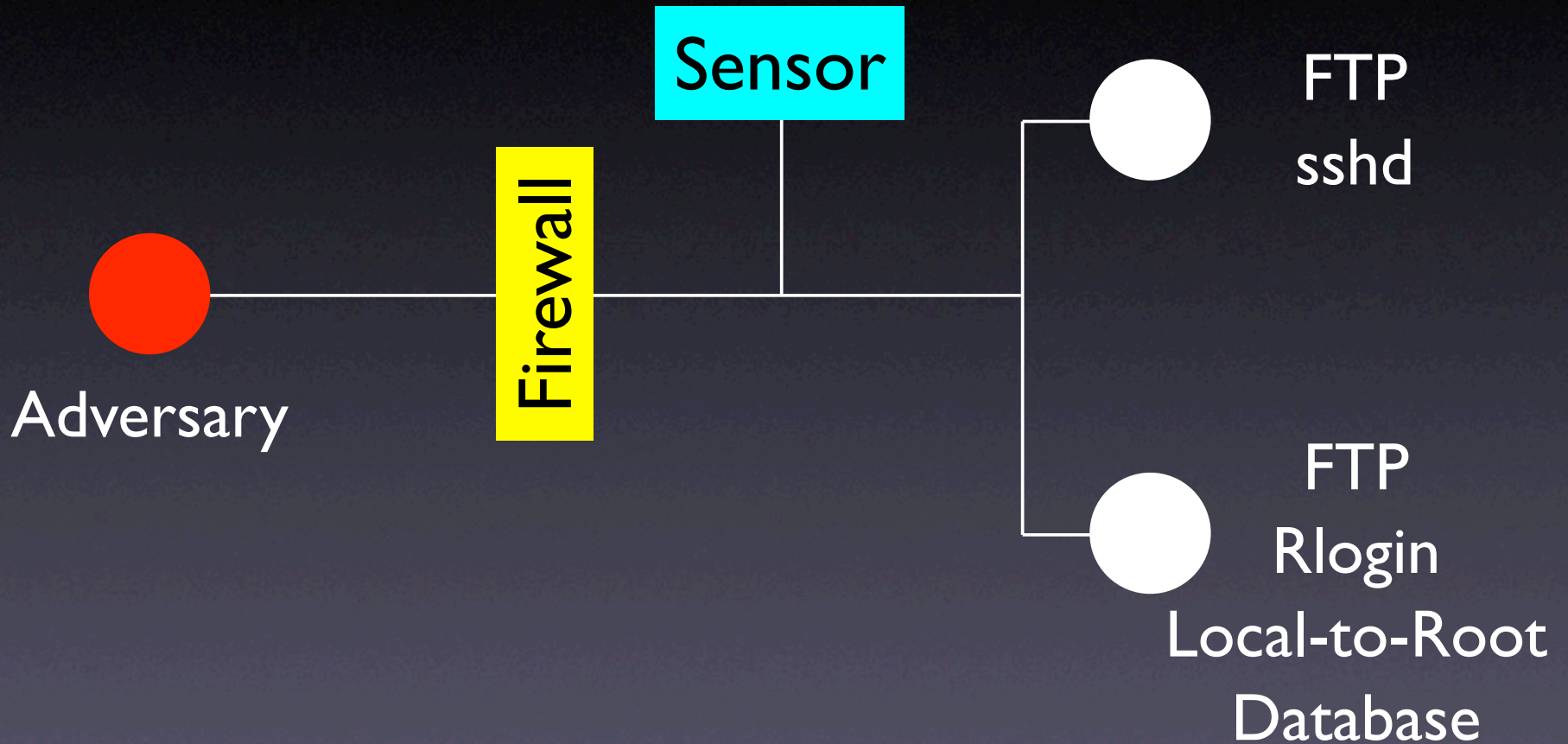
Complete State of the Network



Changing the Network State



Canonical Example



Invariant Condition

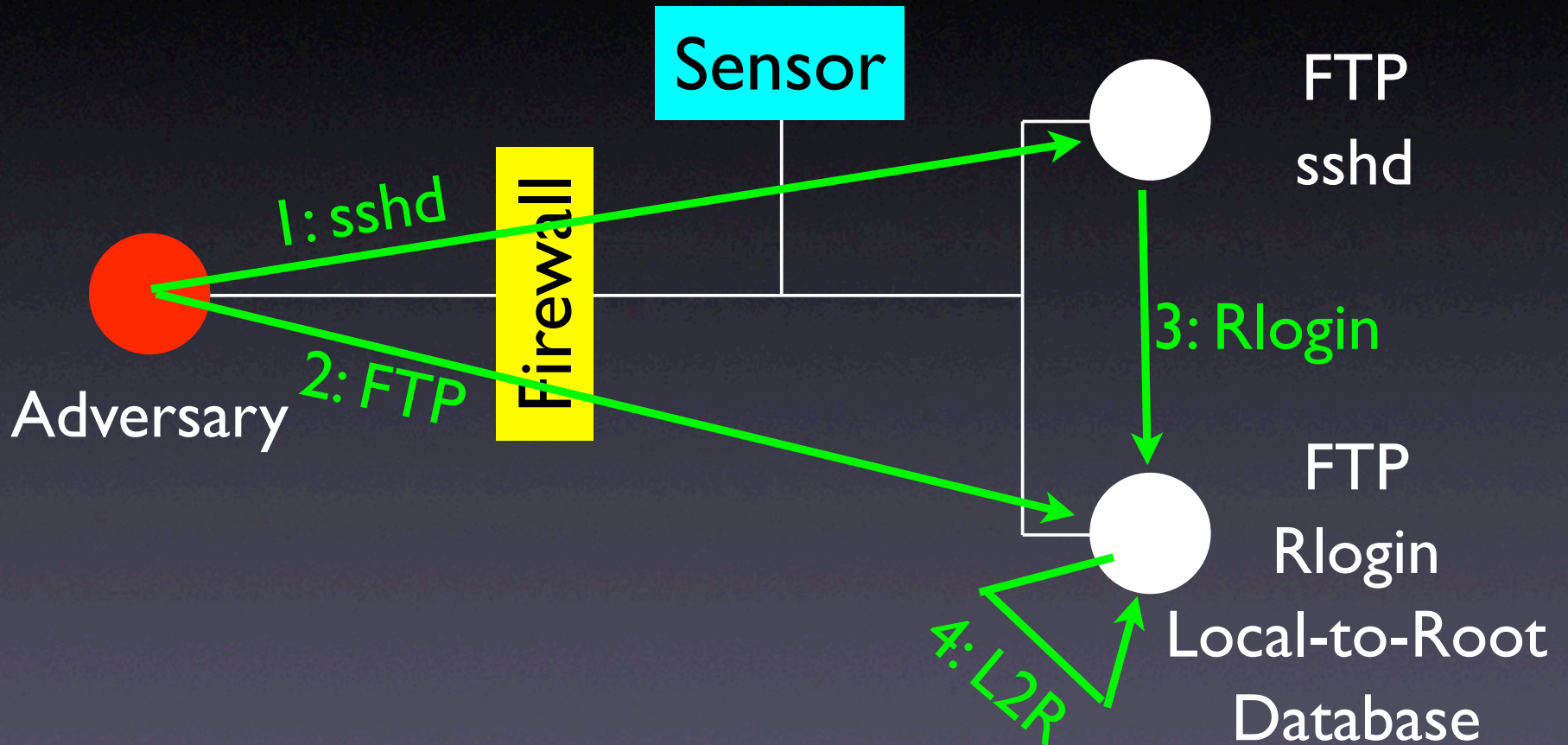
Model's Point of View

AG(network.adversary.privilege[2] < network.priv.root | network.detected)

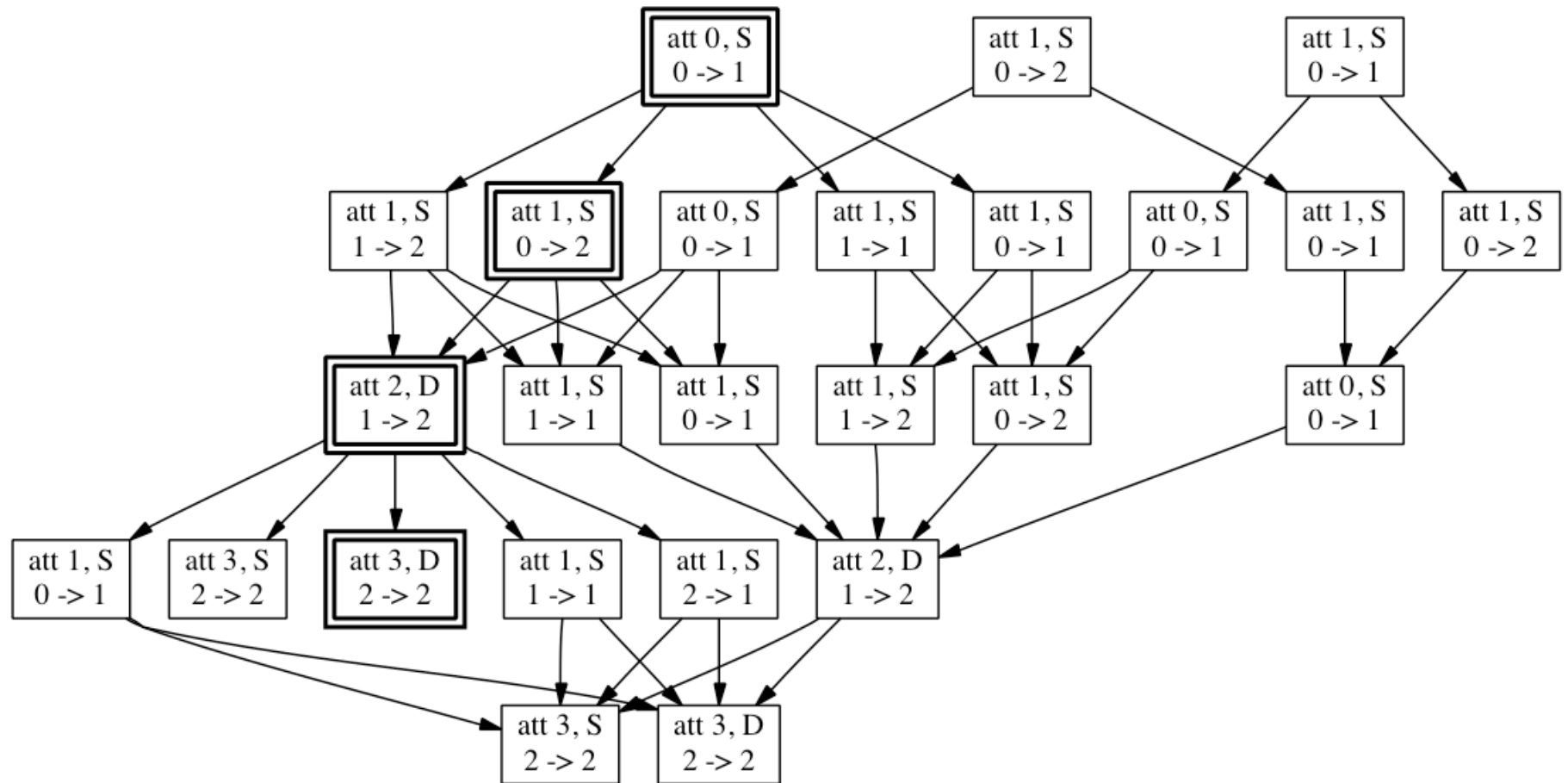
Adversary's Point of View

network.adversary.privilege[2] >=
network.priv.root & -network.detected)

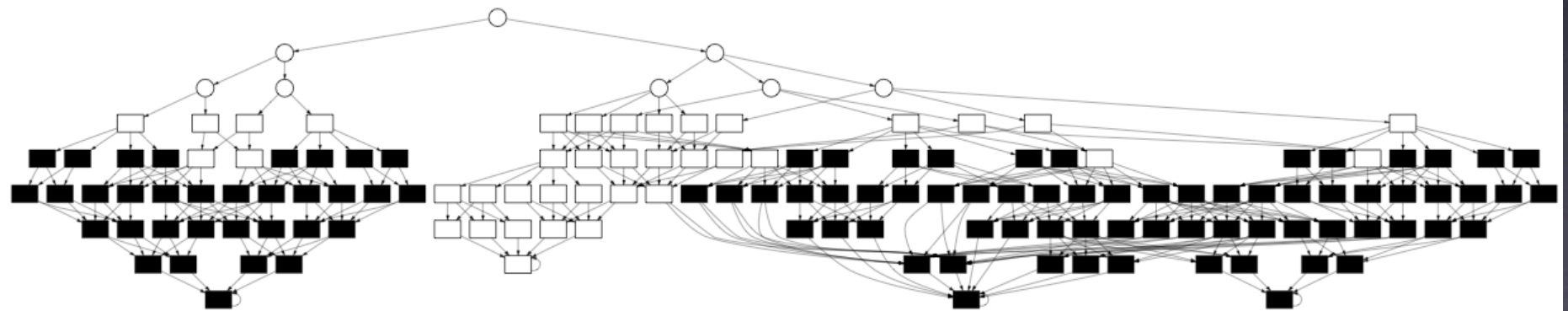
Canonical Example



Attack Graph



Attack Permutations & Detection Avoidance



Statistics for 3 Host Network

- 3 host model has 91 bits of state
- Potentially 2^{91} states (really big)
- Only 101 states are reachable
- 1 Ghz machine needed 5 seconds to build attack graph

Statistics for 5 Host Network

- 5 host model has 229 bits of state
- Potentially 2^{229} states (really really big)
- Only 6190 states are reachable
- 5948 nodes, 68,364 edges
- 1 Ghz machine needed 2 hours to build attack graph

Symbolic Model Verifier Approach

- Benefits
 - Builds on existing engines
 - Recent advances support scaling to large state spaces (possible infinite spaces)
- Formal claims can be made
 - Results are exhaustive - all possible attack paths are identified
 - Results are succinct - only states reachable from the target are identified

Status

- GMU, which used the SMV model checker in one paper published a second paper stating that SMV approach does not scale. Line of work has been abandoned.
- Lincoln Labs, co-author of CMU work, has abandoned SMV in favor of hand-rolled code.
- CMU has abandoned NuSMV, supports “pluggable” model checkers, but is vague on what they are actually using.