Lecture 27

- Evaluating systems
  - SSE-CMM
- Attack trees
- Requires/provides model

SSE-CMM Model

- *Process capability*: range of expected results that can be achieved by following process
  - Predictor of future project outcomes
- *Process performance*: measure of actual results
- *Process maturity*: extent to which a process explicitly defined, managed, measured, controlled, and is effective
- Divides process into 11 areas, and 11 more for project and organizational practices
  - Each process area contains a goal, set of base processes

Process Areas

- Process areas:
  - Administer security controls
  - Assess impact, security risk, threat, vulnerability
  - Build assurance argument
  - Coordinate security
  - Monitor system security posture
  - Provide security input
  - Specify security needs
  - Verify, validate security
- Practices:
  - Ensure quality
  - Manage configuration, project risk
  - Monitor, control technical effort
  - Plan technical effort
  - Define, improve organization's systems engineering process
  - Manage product line evolution
  - Provide ongoing skills, knowledge
  - Coordinate with suppliers

Example: Assess Threat

- Goal: threats to the security of the system will be identified and characterized
- Base processes:
  - Identify natural, man-made threats
  - Identify threat units of measure
  - Assess threat agent capability, threat likelihood
  - Monitor threats and their characteristics

Capability Maturity Levels

- *Performed informally*: perform base processes
- *Planned and tracked*: address project-level definition, planning, performance, verification issues
- *Well-defined*: focus on defining, refining standard practice and coordinating it across organization
- *Quantitatively controlled*: focus on establishing measurable quality goals, objectively managing their performance
- *Continuously improving*: improve organizational capability, process effectiveness

Using the SSE-CMM

- Begin with process area
  - Identify area goals, base processes
  - If all processes present, determine how mature base processes are
    - Assess them against capability maturity levels
    - May require interacting with those who use the base processes
  - Do this for each process area
    - Level of maturity for area is *lowest* level of the base processes for that area
    - Tabular representation (called *Rating Profile*) helps communicate results

Key Points

- First public, widely used evaluation methodology was TCSEC (Orange Book)
  - Criticisms led to research and development of other methodologies
- Evolved into Common Criteria
- Other methodologies used for special environments

Attacks

- Attack trees
- Requires/Provides model
  - JIGSAW attack language

Attack Trees

- Schneier, 1999
  - Similar to fault trees (Amoroso, 1987)
- Methodological approach to describe attacks
  - Also can be used to analyze security

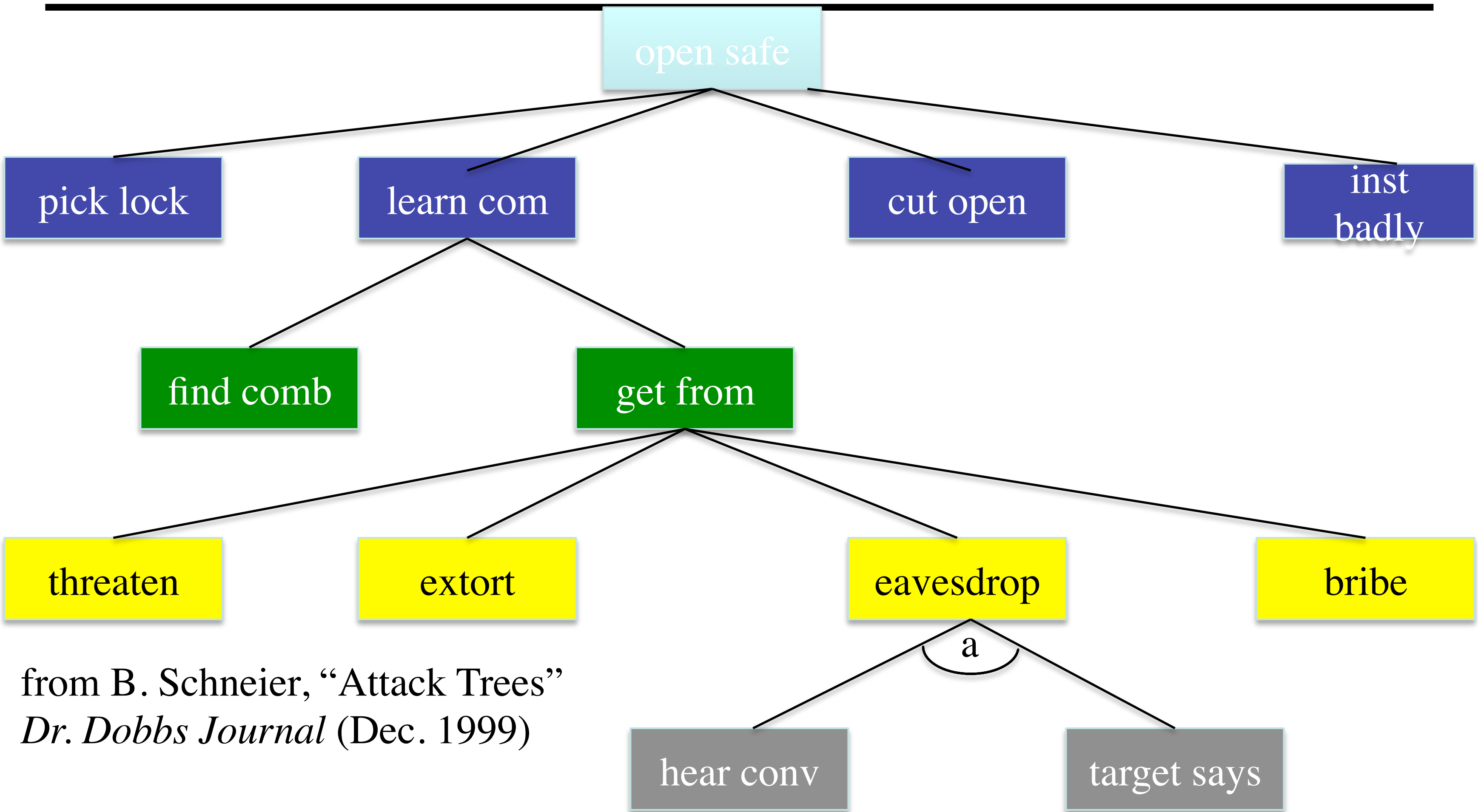
Example

- Goal: open safe
  - Subgoal: pick lock
  - Subgoal: learn combination
  - Subgoal: cut open safe
  - Subgoal: install safe improperly

Subgoal: Learn Combination

- Find written combination
- Get combination from one who knows (target)
  - Threaten
  - Blackmail
  - Eavesdrop
    - Listen to conversation and
    - Get target to state combination
  - Bribe

Attack Tree

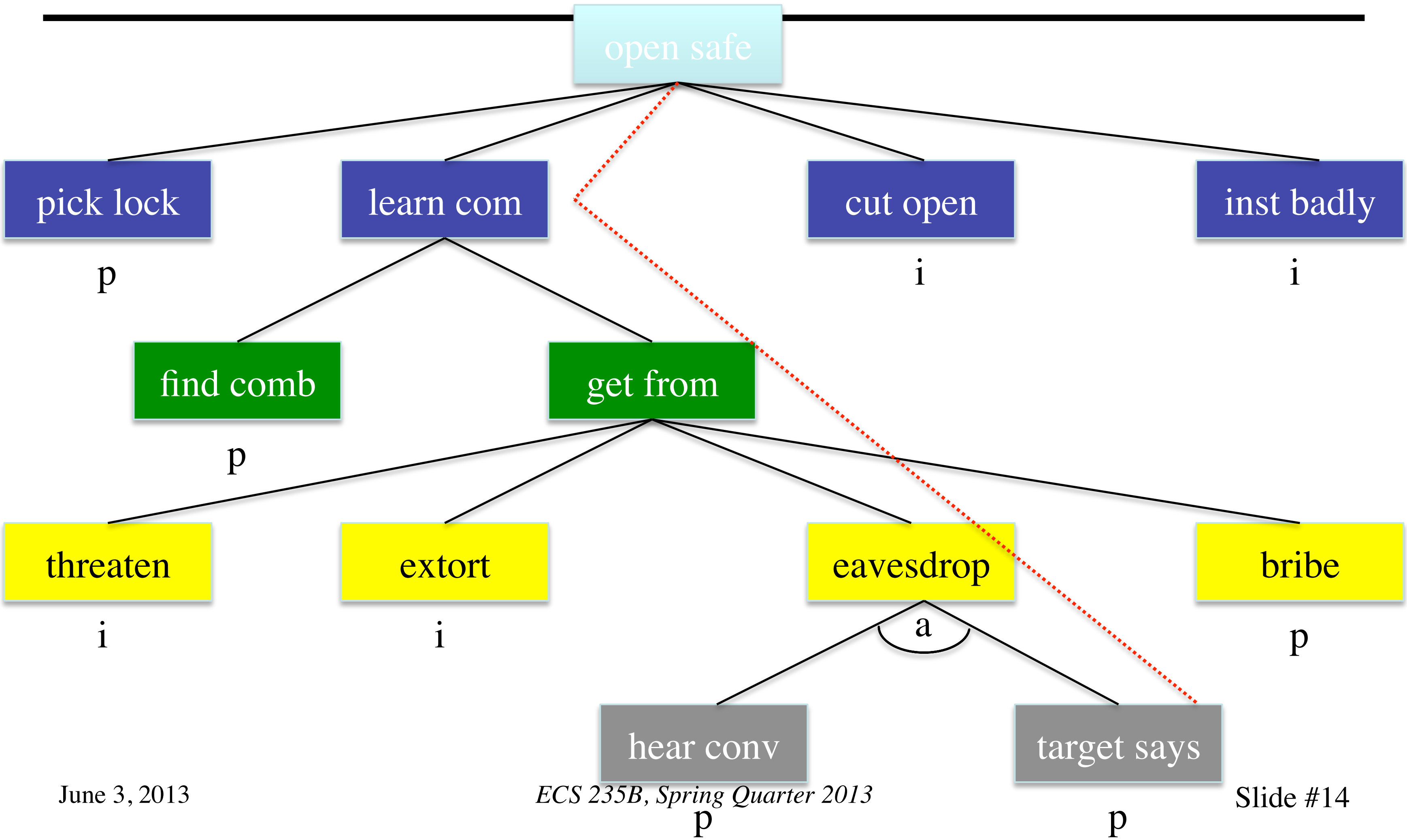


from B. Schneier, “Attack Trees”
Dr. Dobbs Journal (Dec. 1999)

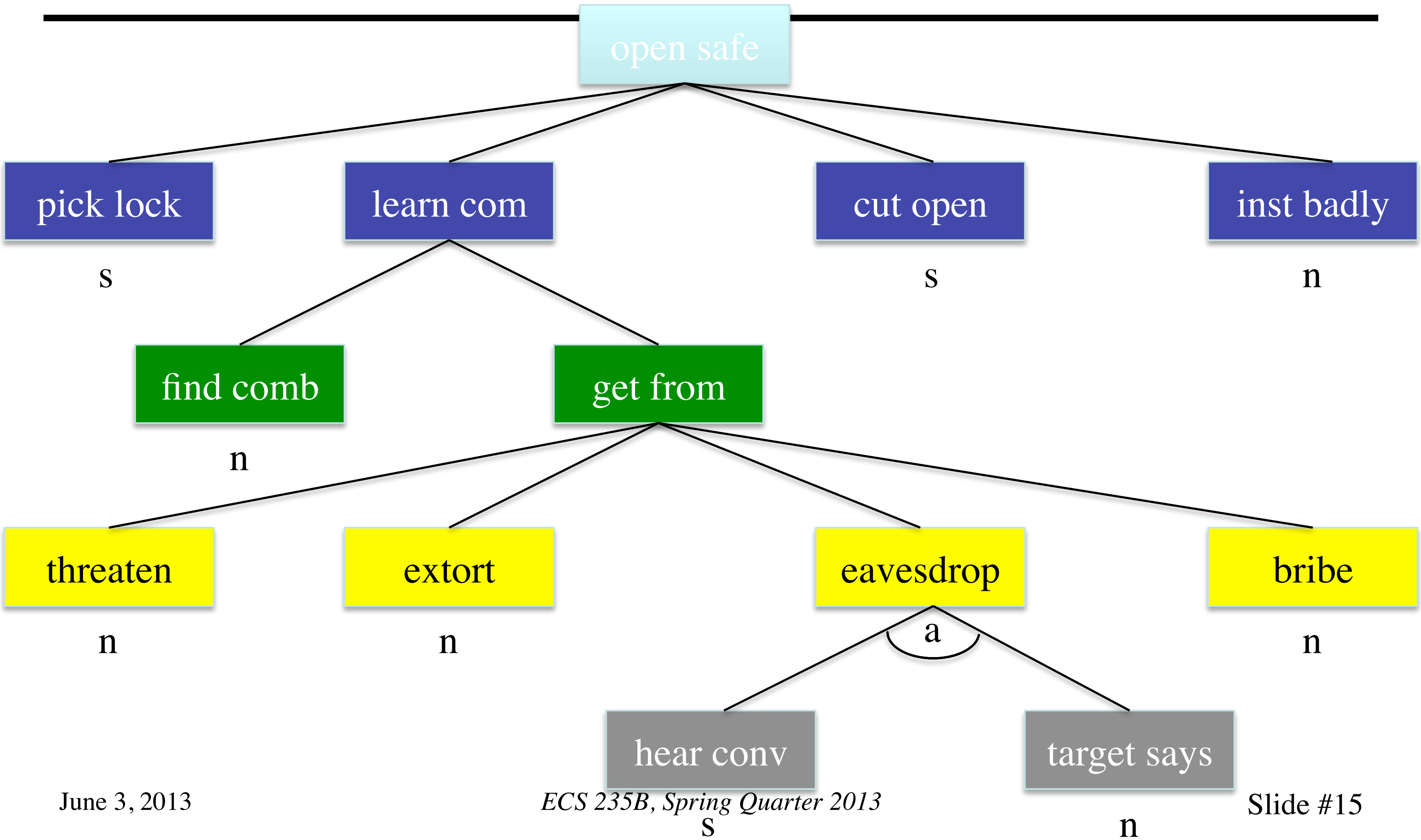
Basic Risk Analysis

- Mark each node with:
  - "p" possible
  - "i" impossible
    - Note these are *estimates*
- Mark each node with:
  - "s" special equipment
  - "n" no special equipment

Attack Tree #2



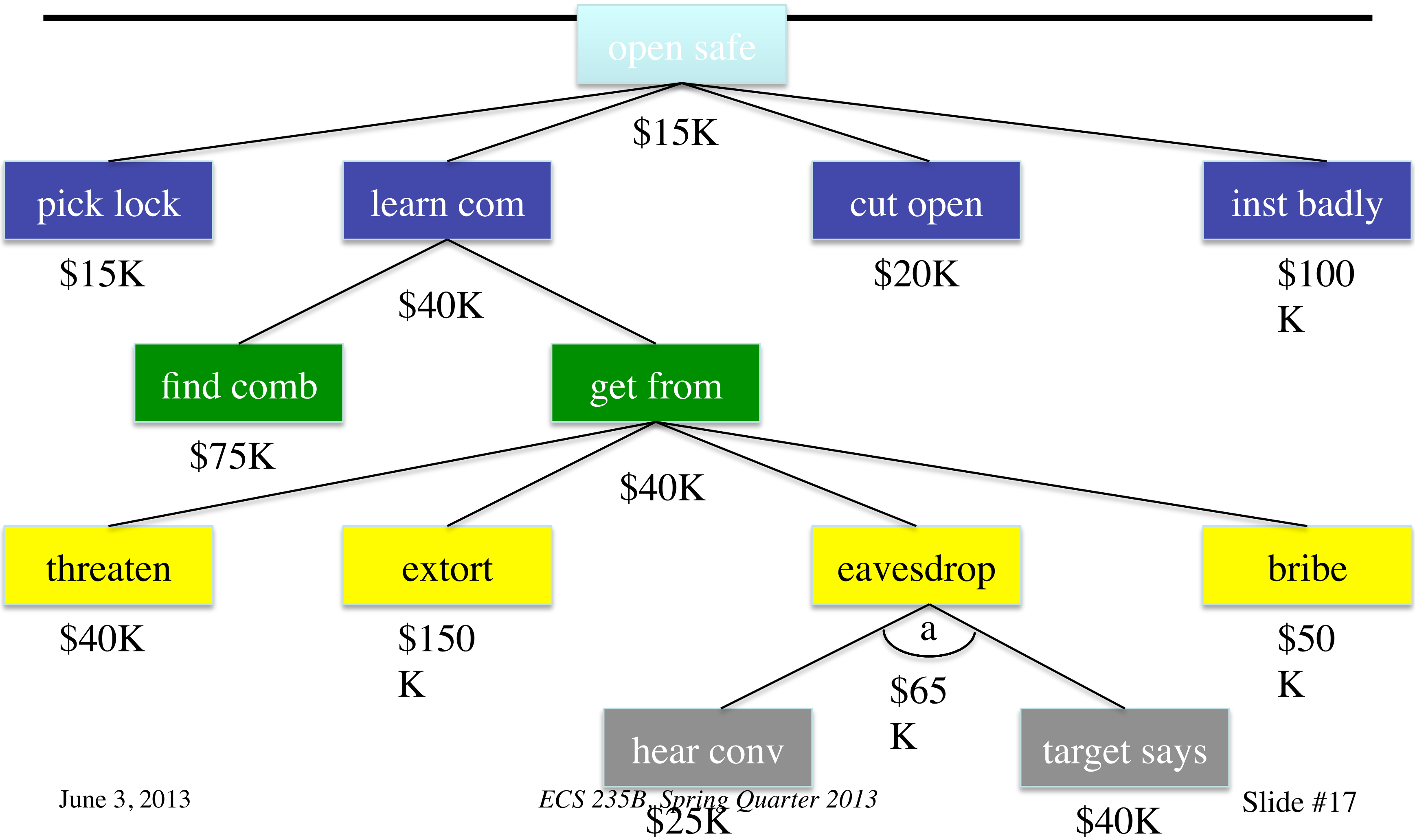
Attack Tree #3



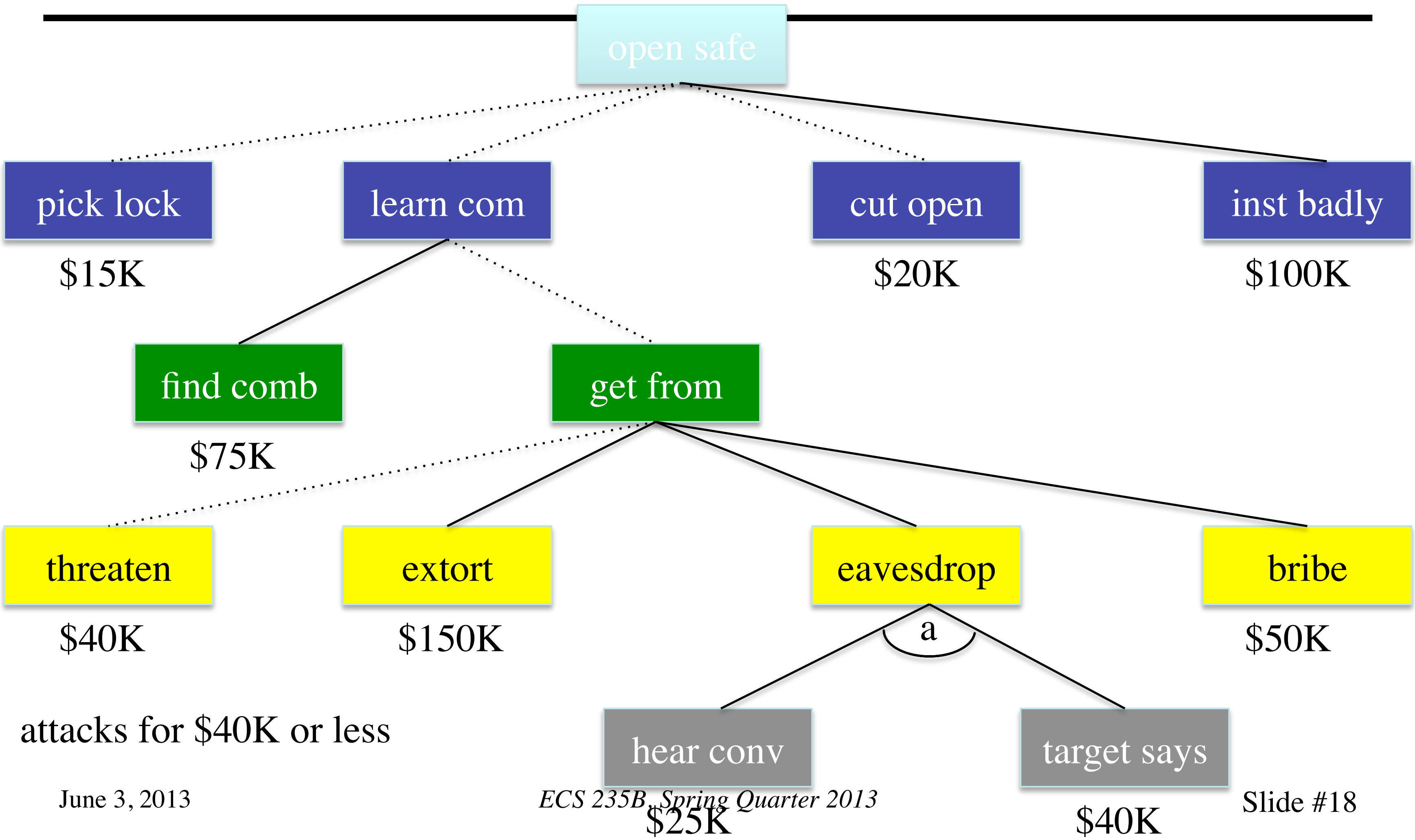
Cost of Attack

- Put costs on endpoints
- Cost of "and" node
  - Sum of costs for child nodes
- Cost of "or" node
  - Minimum of costs of child nodes

Cost of Attacks



Which Are No More Than $40K?

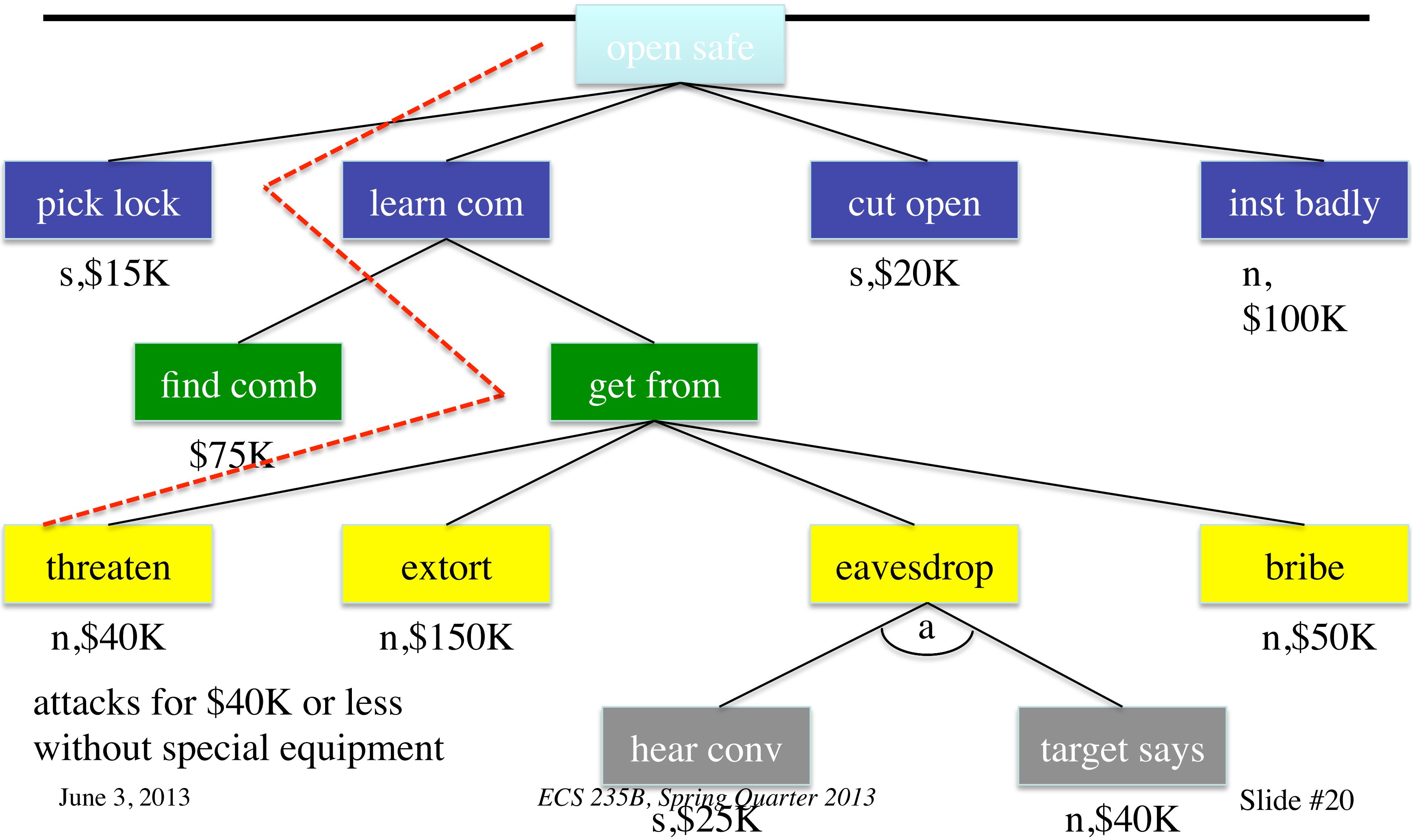


attacks for $40K or less

Combine These

- Which attacks:
  - cost under $40K and
  - require no special equipment?

Combination Tree



attacks for $40K or less
without special equipment

Attacking PGP

Goal: Read a message encrypted with PGP

- Decrypt message (OR)
  - Break asymmetric encryption (OR)
  - Break symmetric encryption
- Determine symmetric key used to encrypt message via other means (OR)
  - Fool sender into encrypting message using public key with known private key (OR)
  - Have recipient sign encrypted symmetric key (OR)
  - Monitor sender's computer memory (OR)
  - Monitor receiver's computer memory (OR)
  - Determine key from pseudorandom number generator (OR
  - Implant malicious logic that sends you the symmetric key
- Get recipient to help decrypt message (OR)
  - Chosen ciphertext attack on public key (OR)
  - Spoof Reply-to or From: field of original message (OR)
  - Read message after it has been decrypted by recipient
- Obtain private key of recipient
  - Factor RSA modulus or calculate ElGamal discrete log (OR)
  - Get private key from recipient's private key ring (OR)
  - Monitor recipient's memory (OR)
  - Implant malicious logic to expose private key (OR)
  - Generate non-secure public/private key pair for recipient

From Schneier, "Attack Trees," *Dr. Dobbs Journal* (Dec. 1999)

Creating Attack Trees

- Identify possible goals
  - Each goal forms separate tree, rooted in higher goal
- Continue iterating until you reach all leaves
  - Good to involve lots of people
- Trees can be reused, as part of larger tree
  - These are, in essence, compartmentalization
- Eminently scalable

Requires/Provides Model

General idea:

- To launch an attack, certain properties must hold
  - These are the *requires* properties
- After the attack, a new set of properties hold
  - These are the *provides* properties
- The "goal" is simply a property

Usual View of Attacks

- Single exploit
  - Goal is *very* short term
  - Violates some part of (implicit) security policy
  - Rarely dangerous
- Sequence of single exploits (*scenario attacks*)
  - Goal is *longer* term, end goal
  - Violates some part of (explicit) security policy
  - Usually dangerous

IDS Languages

- Focus on specific details of exploits
  - Source, destination IP addresses the same
  - Large numbers of TCP SYN packets wit same destination port, address
- Express these in a form that is useful to IDS or other analysis tool
  - CISL, Common Intrusion Specification Language
  - IDS-specific signatures, languages

Issues

- Advantages
  - Tailored for particular IDS or function
    - Such as interchanges among IDSes
  - Express *very* low-level details
- Disadvantages
  - Single exploits
    - Generally do not allow combining attacks
  - Correlation difficult

Problem

- How do we correlate these single attacks into scenario attacks?
  - Example scenario attack
  - Capabilities and concepts
  - The language, JIGSAW
  - Applications

Example: *rsh* Connection Spoofing

Attack scenario:

1. bialystock synfloods franz
2. bialystock probes kafka for starting TCP sequence number information
3. bialystock sends spoofed SYN packet (purportedly from franz) to kafka
4. kafka sends ACK packet to franz, but franz never sees it
5. bialystock sends spoofed packets (purportedly from franz) to kafka, which kafka then executes, as it trusts franz—and attack succeeds

Analysis

- Goal: get kafka to execute commands from untrusted host bialystock
- Subgoal: get kafka to believe trusted host franz is sending the commands
  - Must prevent ACK from kafka from reaching franz
  - Must determine what sequence number kafka would use, so bialystock can use that in “response” to blocked ACK

Blocking Franz

- Used synflood to prevent ACK from reaching franz
- Could have used *anything* that would prevent such reception
  - packet storm attack, saturating network
  - cutting wires
  - ping-of-death to get franz to hang
  - lots more …

Other Variants

- Distribution
  - Use bialystock to send command
  - Use bloom for synflood
    - And you can forge the source IP address in those packets ...
- Resequencing
  - Start the probing for sequence number *before* the synflood is launched

Requires/Provides Model

- Capabilities
  - Information, situation required for attack to succeed
    - User login: requires access, user name, password; system requires access to password validation database
  - May represent “links” (lines)
  - May represent leaves
    - Encapsulate assumptions external to analysis

More About Capabilities

- Inherent implication
  - kafka can’t send ACK packets to franz
  - franz can’t receive ACK packets from kafka
    - Either implies the other
    - These may need to be stated explicitly, but you can automate their generation if needed

Requires/Provides Model

- Concepts
  - Situations defining subtasks in scenario attacks
  - Defines requirements for concept to hold
    - Boolean relations on capabilities, configurations
  - Idea: if capabilities satisfy requirements, concept gives new capabilities

Model Features

- Multiple events can produce equivalent capabilities
  - Reason in terms of *effects* of attack (capabilities produced), not *what* the attack is
- Attack scenarios may have many variants
  - Again, focus on capabilities produced
- Exploits can be combined in unknown ways to create sophisticated attacks
  - But they will all produce capabilities

More Features

- Attacks compose based on provided/required capabilities
  - In essence, capabilities for the “edges” of the attack graph
- Known exploits/actions/vulnerabilities form terminals in the model
  - This is simply a convenience
- Attacks can be defined locally without knowing how they will be used

JIGSAW

- Language to specify model
- Capability templates
  - Capability specification: named collection of typed attribute-value pairs
- Concepts
  - Set of required capabilities

Example: Capability

```
capability xcap is
  ip_addr:        ip_addr_type;
  port_set:   set of port;
  start_time: time_type;
  end_time:   time_type;
end.
```

Example: Concept

```
concept RSH_Connection_Spoofing is
  requires
    Trusted_Partner:     TP;
    Service_Active:      SA;
    Prevent_Packet_Send: PPS;
    extern SeqNumProbe:  SNP;
    Forged_Packet_Send:  FPS;
  with
    TP.service is RSH,              # Service in trust relation is RSH
    PPS.host is TP.trusted,         # Blocked host is trusted partner
    FPS.dst.host is TP.trustor,     # Spoofed packets to trustor
    SNP.dst.host is TP.trustor,     # Probed host is trustor
    FPS.src is [ND.host, PPS.port], # Claimed source of forged packets blocked
    SNP.dst is [SA.host, SA.port],  # Probed host running RSH on normal port
    SA.port is TCP/RSH,
    SA.service is RSH,
    SNP.dst is FPS.dst              # Probed host is where packets are sent
    active(FPS) during active(PPS)  # Forged packets sent while DoS attack
  active
  end;
```

Example: Concept (*con't*)

```
concept RSH_Connection_Spoofing is
  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 RSH server
    REX.to <- FPS.dst,
    REX.using <- RSH,
  end;
  action
    true -> report("RSH Connection Spoofing: got TP.hostname!")
  end;
end;
```

Neat Aspects

- Using IDS alerts to drive capability sets
  - IDS does the gathering of data for the "leaves"; JIGSAW (or other framework) does extrapolation
- Extensions
  - Can "pre-package" attacks
  - Can easily add capabilities
    - Either directly or through inferencing chains derived from inherited implications

More Neat Things

- Predictions
  - Can state what capabilities exist as a result of actions, and from concepts see what attacks are possible
- Automated response
  - Goal: remove capability to thwart potential attack
  - Do this automatically by changing system configuration or data driving attack

Work Built on This

- J. Zhou *et al*.: Model to correlate ID alerts for networks to detect high-level attacks
- S. Peisert *et al*.: Model to direct forensic analysis based on goals of attack