The Atomizer: Support for Security Management Infrastructure

Joshua D. Guttman
John D. Ramsdell
Charles M. Schmidt
F. Javier Thayer Fábrega

September 1998

MITRE

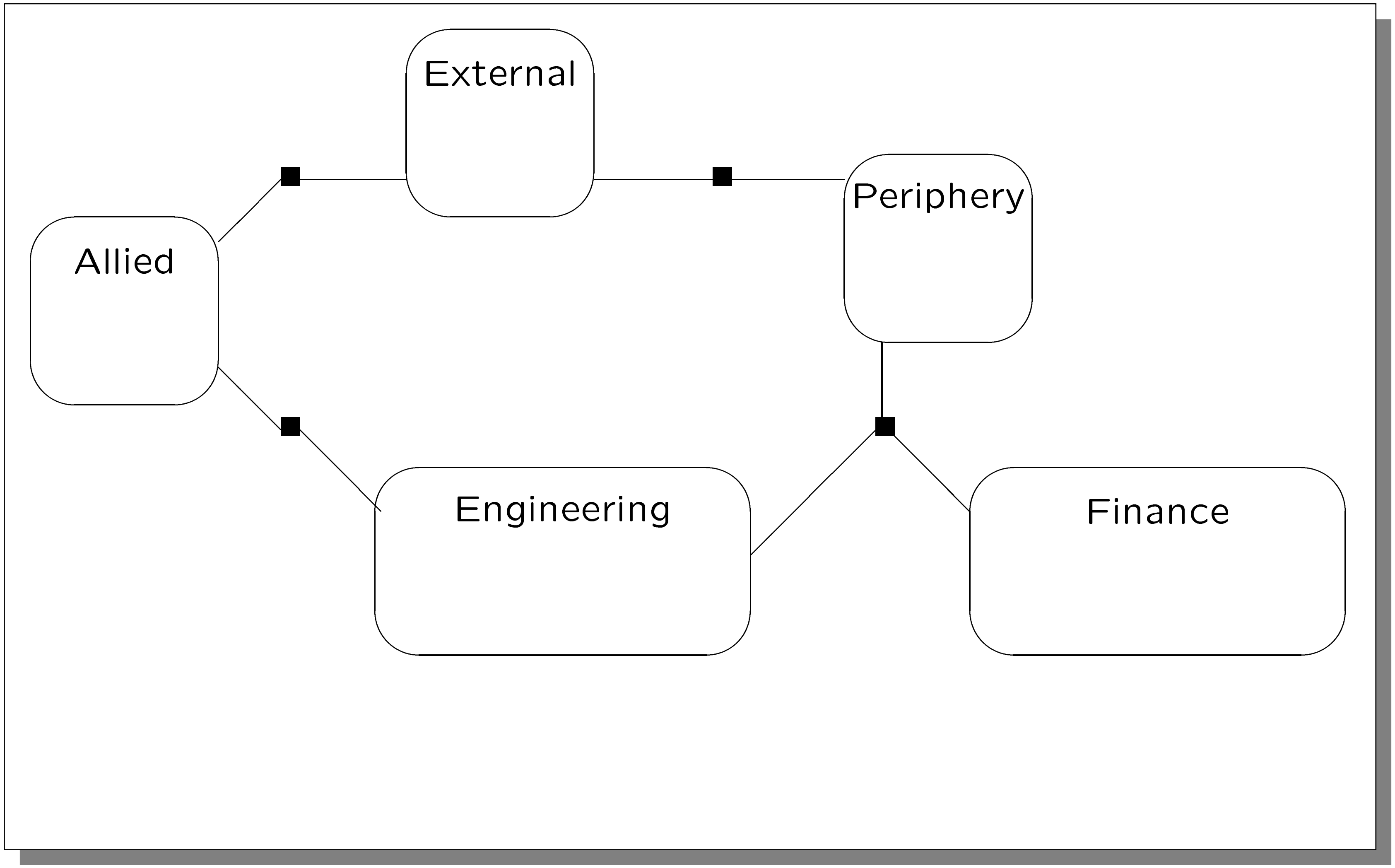
Problem: Managing Filtering Routers

- Filtering routers: Basic part of network security
  - Organizations have many routers doing filtering
  - Configurations are complicated and evolving
  - Multiple network paths
- The top-down approach
  - Determine security-relevant routers, areas they define
  - Define end-to-end packet access-control policies
  - Calculate filtering behavior to enforce policies
  - Implemented in Network Policy Tool (NPT), FY 96–97
- The bottom-up approach
  - Discover existing policy
  - Preserve most existing filter files
  - Make changes to policy or filters as needed

Starting Point: NPT Framework

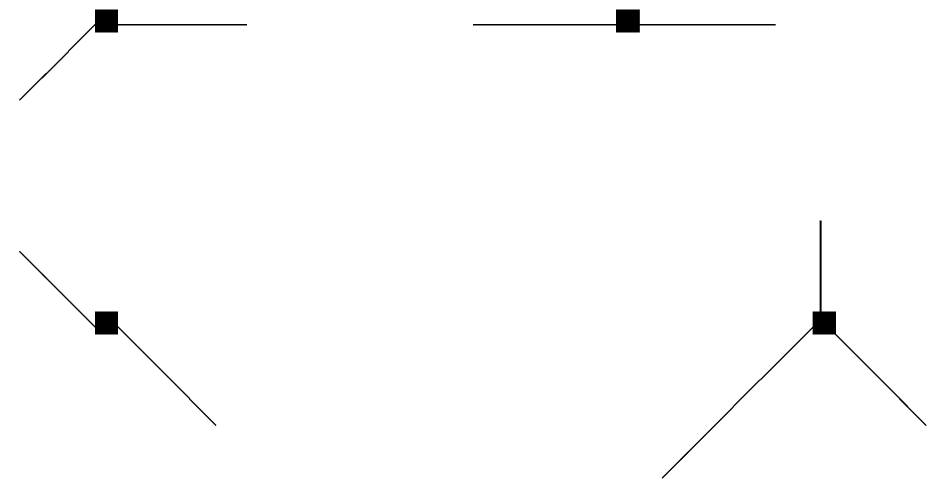
- Two key ideas in NPT:
  - Regard net as bipartite graph
  - Use symbolic computation on "abstract packets"
- Bipartite graph
  - Must know security-relevant routers
  - Can derive areas by network mapping
  - Edges mean "has an interface on"
  - Motivation: Graphs small, computations tractable
- Symbolic computation on abstract packets
  - Source & destination addresses & ports vary, but it doesn't make a difference
  - Symbolic computation: Boolean operations on sets

A Network Graph



MITRE

Deriving a Network Graph



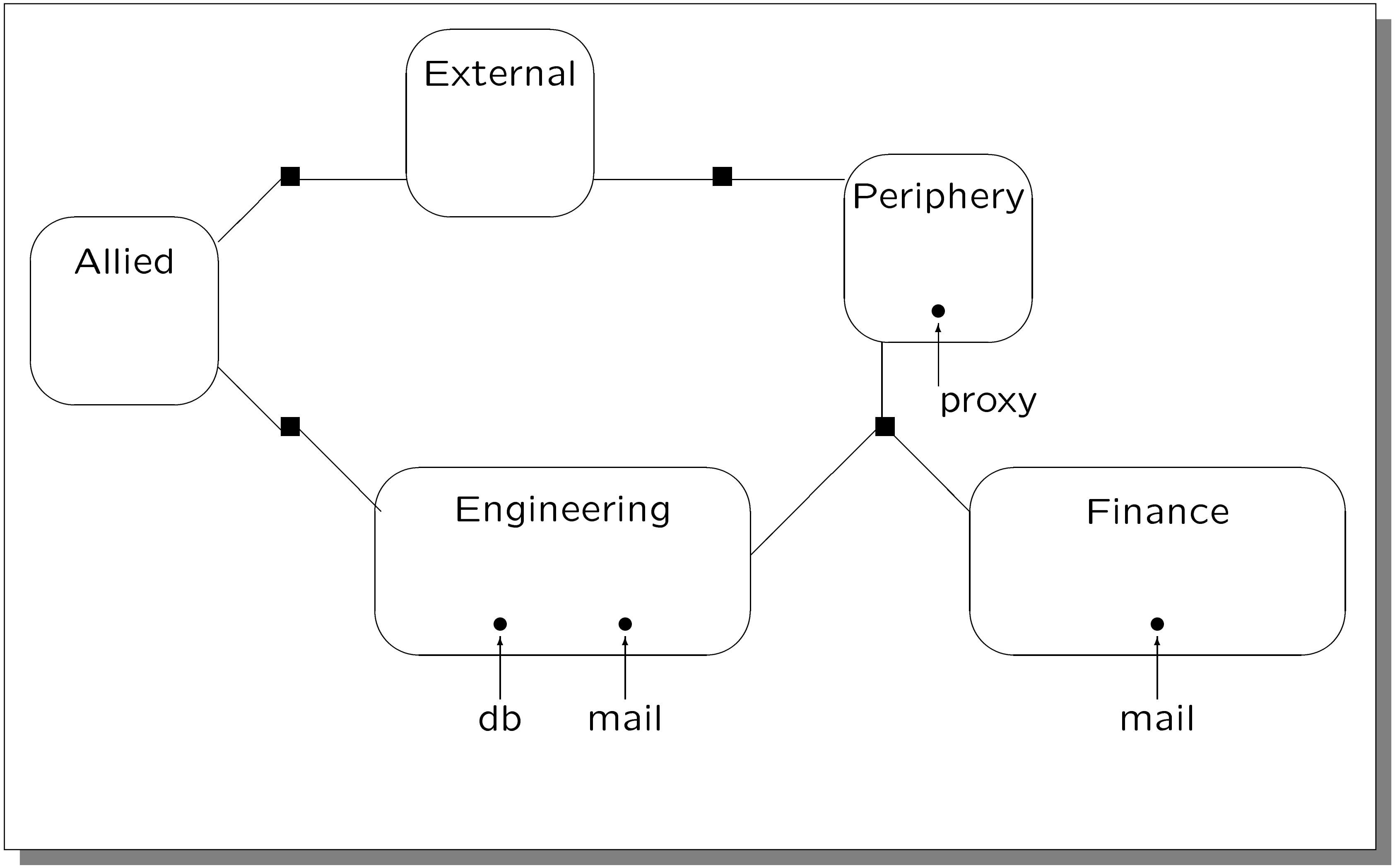
MITRE

Abstract Packets and Real Packets

- Given some filters, only certain aspects count about source/destination address fields and protocol, port fields
- Addresses i, j may be clustered into equivalence classes
  - $i \sim j$ if $p[i]$ and $p[j]$ always treated the same
  - If filters give same outcome, don't distinguish i, j
  - An abstract address is an equivalence class
  - Can often guess right choices;
    can then use top-down approach
  - Similar idea applies to abstract services (ports etc.)
- Abstract packet:
  - Tuple of abstract source, destination, service
  - Abstract packet defines little box in
    3-d space of concrete packets

MITRE

A Network Graph with Hosts



Nine Abstract Addresses

- external ordinary
- allied ordinary
- periphery ordinary, proxy
- financial ordinary, financial mail server
- engineering ordinary,
  engineering mail server, engineering db server

The Security Management Problem

- Organizations have filters in place
  - Results of complicated evolution
  - Encode some security goals
  - Adapted to meet service needs
- What security policy do the filters add up to?
  - NPT can help: Same graph traversals are relevant
- But: What are the abstract addresses?
  - What changes of address make a difference?
  - Abstract address are sets;
    interchanging members of same set never
    makes a failing packet pass (or vice versa)
- How to deduce abstract addresses
  from the configuration files themselves?

MITRE

Router Configuration File (e.g. Cisco)

- Access list entries permit or deny based on
  - Source and destination address (using netmasks)
  - Protocol and protocol-specific properties
    - For TCP and UDP, ports and ACK bits
    - For ICMP and IGMP, message type and code
- Each line defines a triple of sets:
  - Matching source addresses
  - Matching destination addresses
  - Same service: protocol and specific attributes

First match determines disposition (permit/deny)

MITRE

Example Config File

```
access-list 101 permit ip host 129.83.10.1 \
                          129.83.114.0 0.0.0.255
access-list 101 permit ip host 129.83.11.1 \
                          129.83.114.0 0.0.0.255
access-list 101 deny ip host 129.83.10.1 any
access-list 101 deny ip host 129.83.11.1 any
access-list 101 permit ip 129.83.10.0  0.0.0.255  \
                          129.83.115.0 0.0.0.255
access-list 101 permit ip 129.83.11.0  0.0.0.255  \
                          129.83.115.0 0.0.0.255
```

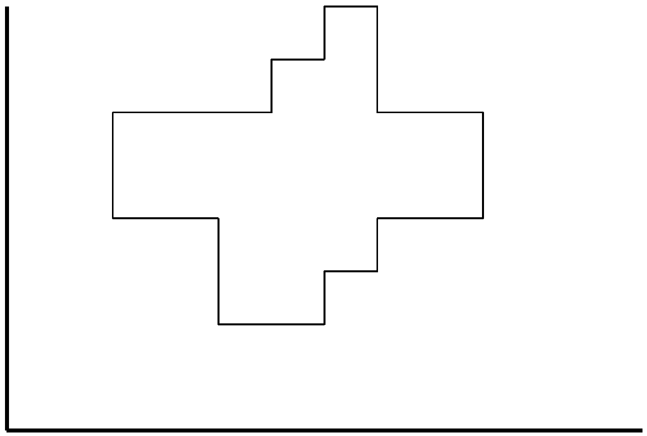
- **Two hosts can talk to 129.83.114.***
- **Others on their networks can talk to 129.83.115.***
- **Five sets of relevantly different addresses**

```
129.83.114.*                 129.83.115.*
129.83.{10,11}.1             129.83.{10,11}.* - 1
[everything else]
```

MITRE

Product Boolean Algebras (2-D)

Given a set E, how can we choose atoms s_i, d_i so that E is representable in the form $E = \bigcup_I s_i \times d_i$



A source atom contains
those sources that share a dest-section

MITRE

Many Config Files

- What we read off one config file is only part of the story
  - The real abstract addresses can be finer, not coarser
  - If A_k is one filter, $i \sim_{k,src} j$ iff
    $\forall x, y \,.\, \langle i, x, y\rangle \in A_k \equiv \langle j, x, y\rangle \in A_k$
  - Source pre-atoms of A_k: equivalence classes of $\sim_{k,src}$
  - Combining families of pre-atoms is easy
- The whole story: finer than $\sim_{k,src}$, but compatible
  - If $A_1 \ldots A_n$ are the filters, $i \sim_{src} j$ iff
    $\forall A_k, x, y \,.\, \langle i, x, y\rangle \in A_k \equiv \langle j, x, y\rangle \in A_k$
- Source atoms: the equivalence classes of $\sim_{src}$
  - Any relevant set of sources is a union of source atoms

MITRE

Combining Pre-Atom Families

```
let superimpose b1 b2 =
      if        B.null_intersection b1 b2
      then      No_overlap
      else      Overlap ((B.conjoin b1 b2),
                 (B.difference b1 b2),
                 (B.difference b2 b1))

let rec add_pre_atom b = function
      []                 -> [ b ]
    | old_atom :: rest   ->
        match (superimpose b old_atom) with
          No_overlap    -> old_atom :: (add_pre_atom b rest)
        | Overlap (common, new_shard, old_shard)
          -> common :: (old_shard :: (add_pre_atom new_shard rest))

let rec combine_pre_atom_lists = function
    []                         -> (fun ys -> ys)
  | x :: rest                  -> fun ys ->
      add_pre_atom x (combine_pre_atom_lists rest ys)
```

MITRE

Processing an Config File

- Parse file: construct lists of permit/deny BDD triples
- Combine lines for each access list
  Represents the set of packets that pass that filter:
  - For permit line: Either this line applies or . . .
  - For deny line: This line does not apply and . . .

  where the ". . ." represents the remainder of the access list
- Need a representation of sets of packets
  and sets of sources, destinations, services
  - An address is a bit string
  - A set of addresses is a boolean function of bit strings

MITRE

BDDs

- Reduced, ordered Binary Decision Diagrams (BDDS)
  - Binary, acyclic graphs with two sinks (true and false)
  - Bits examined in fixed order
  - Same boolean function occurs only once in memory
- Representation tuned for the sets of bit strings that occur naturally
  - Bits are relevant in a definite order
  - Low order bits may not count
  - Operations do not “mix” distant bits
- Efficient implementation strategies well understood

A Sample BDD

Source Atoms for Example

```
access-list 101 permit ip host 129.83.10.1 129.83.114.0 0.0.0.255
access-list 101 permit ip host 129.83.11.1 129.83.114.0 0.0.0.255
access-list 101 deny ip host 129.83.10.1 any
access-list 101 deny ip host 129.83.11.1 any
access-list 101 permit ip 129.83.10.0  0.0.0.255 \
                          129.83.115.0 0.0.0.255
access-list 101 permit ip 129.83.11.0  0.0.0.255 \
                          129.83.115.0 0.0.0.255

[Include 129.83.10.1 255.255.254.255
 Exclude 0.0.0.0 0.0.0.0
  ;
 Exclude 129.83.10.0 255.255.254.0
 Include 0.0.0.0 0.0.0.0
  ;
 Exclude 129.83.10.1 255.255.254.255
 Include 129.83.10.0 255.255.254.0
 Exclude 0.0.0.0 0.0.0.0  ]
```

Representing a Filter

```
Or(
  And((Source: Exclude 129.83.10.1 255.255.254.255
               Include 129.83.10.0 255.255.254.0
               Exclude 0.0.0.0 0.0.0.0)
      (Destination: Include 129.83.115.0 255.255.255.0
                    Exclude 0.0.0.0 0.0.0.0))
  And((Source: Include 129.83.10.1 255.255.254.255
               Exclude 0.0.0.0 0.0.0.0)
      (Destination: Include 129.83.114.0 255.255.255.0
                    Exclude 0.0.0.0 0.0.0.0)))
```

Status

- Machinery well tested and highly efficient
  - Processes test files correctly
  - Processes large, real files quickly
    E.g. 1345 lines in 43 seconds, 65MB (Pentium II)
  - Not yet emitting NPT files
- Very small amount of programming remaining
- Will write up report or conference paper

Conclusions

- Efficient method to determine atoms of boolean algebras
- Probably applicable to other problems
- Key to automating security management for filtering routers

MITRE