Chapter 13
Security and Trust

Professor Hossein Saiedian
EECS818: Software Architecture
University of Kansas
Outline of Chapter 13

1. Security
2. Design Principles
3. Architectural Access Control
4. Trust Management
5. End Matter
Objective of Chapter 13

- Introduction of the different aspects of security, including confidentiality, integrity, and availability
  - Discuss several general design principles of security
  - Provide a technique for architectural access control
  - Conclude with presentation of an architectural approach for constructing trust enabled decentralized applications
13.1 Security [1/5]

• The National Institute of Standards and Technology (NIST) defines computer security as
  – “The protection afforded to an automated information system in order to attain the applicable objectives of preserving the integrity, availability, and confidentiality of information system resources (includes hardware, software, firmware, information/data, and telecommunications)”

• Three main aspects of security: confidentiality, integrity and availability
13.1 Security [2/5]

- Confidentiality
  - Preserving the confidentiality of information means preventing unauthorized parties from accessing the information
  - Also referred to as secrecy
  - Applying this concept to software architecture
    - Software systems should take proper measures while exchanging information to protect confidential information from being intercepted by rogue parties
    - Software systems should store sensitive data in a secure way
13.1 Security [3/5]

• Integrity
  – Maintaining the integrity means that only authorized parties can manipulate the information and do so only in authorized ways
  – Analogous constructs exist at the programming language level
    • Example: in Java, member fields and functions are given access levels as private, package, protected and public
13.1 Security [4/5]

• Integrity [continued]
  – At the software component and architectural level, similar protective mechanisms can be applied to the interfaces of components
    • Such interfaces should be designated and separated from the others and receive more scrutiny during design
  – Depending on security requirements, different levels of authentication may be used for software components and connecters
13.1 Security [5/5]

- **Availability**
  - Resources are *available* if they are accessible by authorized parties on all appropriate occasions.
  - In contrast, if a system cannot deliver its services to its authorized users because of the activities of malicious user then its services are unavailable.
    - It is said to be the victim of a denial of service (DoS) attack.
13.2 Design Principles [1/11]

• Security aspects of software systems should be considered from project’s start
  – During system conception the security requirements should be identified and corresponding security measures designed
  – Patching a security problem after system is built can be prohibitively expensive

• Security requirements also evolve with other requirements
  – Architect should anticipate possible changes and design flexibility into the security architecture
13.2 Design Principles [2/11]

• Principle of Least Privilege
  – The principle of least privilege states that: A subject should be given only those privileges it needs to complete its task
  – The rationale is that even if a subject is compromised, the attacker has access only to a limited set of privileges
  – Software architecture makes it easier to determine the least privileges components
    • Explicit models of architecture enable analysis of communication and control paths to determine the necessary attributes
13.2 Design Principles [3/11]

- **Principle of Fail-Safe Defaults**
  - The principle of fail-safe defaults states that unless a subject is granted explicit access to an object, it should be denied access to that object.

- **Principle of Economy of Mechanism**
  - The principle of economy of mechanism states that security mechanisms should be as simple as possible – also referred to as the KISS (Keep it Simple and Small) principle.
  - Complexity is the enemy of security.
13.2 Design Principles [5/11]

- Principle of Complete Mediation
  - The principle of complete mediation requires that all accesses to entities be checked to ensure that they are allowed, irrespective of who is accessing that
  - Applying this principle to a software system requires all communication to be checked to thoroughly
  - The principle of economy of mechanism helps achieve complete mediation
13.2 Design Principles [6/11]

• Principle of Open Design
  – States that the security of a mechanism should not depend upon the secrecy of its design or implementation
    • Prefer known communication protocols vs proprietary ones
  – Revealing the internals of a system can actually increase its security
  – Furthermore, during its operation and evolution phases, the system can be studied and refined accordingly to make it more secure
13.2 Design Principles [7/11]

- Principle of Separation of Privilege
  - The principle of separation of privilege states that a system should not grant permission based on a single condition
    - Sensitive operations should require the cooperation of more than one key party
  - Software architecture descriptions facilitate the checking of this principle
    - If a component possesses multiple privileges, redesign it
13.2 Design Principles [8/11]

• Principle of Least Common Mechanism
  – The principle of least common mechanism states that mechanisms used to access separate resources should not be shared (to avoid cross-contamination)
  – In the context of software architecture, this implies the need for careful scrutiny when certain software architectural styles (e.g., a blackboard) are used
13.2 Design Principles [9/11]

• Principle of Psychological Acceptability
  – Security mechanisms should not make the resource more difficult to access for legitimate users than if the security mechanisms were not present
  – Likewise, the human interface of security mechanisms should be designed to match the mental model of the users and should be easily usable
  – It is important to design security mechanisms keeping user’s psychological acceptability in mind
13.2 Design Principles [10/11]

• Principle of Defense in Depth
  – States that a system should have multiple defensive countermeasures to discourage potential attackers
  – Requires each component in a path that leads to critical components to implement proper security measures in its own context
  – Example: Microsoft Internet Information services (IIS) Version 6 handles WebDAV requests
### 13.2 Design Principles [11/11]

- **Security for Microsoft IIS**

<table>
<thead>
<tr>
<th>POTENTIAL PROBLEM</th>
<th>PROTECTION MECHANISM</th>
<th>DESIGN PRINCIPLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>The underlying dll (ntdll.dll) was not vulnerable because...</td>
<td>Code was made more conservative during the Security Push.</td>
<td>Check precondition</td>
</tr>
<tr>
<td>Even if it were vulnerable...</td>
<td>Internet Information Services (IIS) 6.0 is not running by default on Windows Server 2003.</td>
<td>Secure by default</td>
</tr>
<tr>
<td>Even if it were running...</td>
<td>IIS 6.0 does not have WebDAV enabled by default.</td>
<td>Secure by default</td>
</tr>
<tr>
<td>Even if Web-based Distributed Authoring and Versioning (WebDAV) had been enabled...</td>
<td>The maximum URL length in IIS 6.0 is 16 Kbytes by default (&gt;64 Kbytes needed for the exploit).</td>
<td>Tighten precondition, secure by default</td>
</tr>
<tr>
<td>Even if the buffer were large enough...</td>
<td>The process halts rather than executes malicious code due to buffer-overflow detection code inserted by the compiler.</td>
<td>Tighten postcondition, check precondition</td>
</tr>
<tr>
<td>Even if there were an exploitable buffer overrun...</td>
<td>It would have occurred in w2wp.exe, which is running as a network service (rather than as administrator).</td>
<td>Least privilege (Data courtesy of David Aucsmith)</td>
</tr>
</tbody>
</table>
13.3 Architectural Access Control

13.3.1 Access Control Models [1/4]

- A most basic security mechanism used to enforce secure access is a *reference monitor*
  - Controls access to protected resources
  - Decides whether access should be granted or denied
- Two dominant types of access control models are
  - Discretionary access control (DAC) models: Access is based on requestor, the accessed resource and whether requestor has permission to access the resource
  - Mandatory access control (MAC) models: Access decision is made according to a policy specified by a central authority
13.3 Architectural Access Control

13.3.1 Access Control Models [2/4]

• Classic Discretionary Access Control
  – The Access Matrix Model is the most commonly used discretionary access control model
  – In this model, a system contains a set of subjects
    • Have privileges and a set of objects on which these privileges can be exercised
    • An access matrix specifies the privilege a subject has on a particular object
  – The access matrix can be implemented directly resulting in an authorization table
13.3 Architectural Access Control
13.3.1 Access Control Models [3/4]

• Role-Based Access Control
  – A role-based access control (RBAC) model is a more recent extension of the classic access control model
    • Extra level of indirection called a role
    • Roles become the entities that are authorized with permissions
  – RBAC allows roles to form a hierarchy
    • A senior role can inherit from a junior role
  – RBAC allows clear specification of the roles that cannot be performed simultaneously by the user
13.3 Architectural Access Control

13.3.1 Access Control Models [4/4]

• Mandatory Access Control
  – Mandatory access control models are less common and more stringent than discretionary models
  – Can prevent both direct and indirect inappropriate access to a resource
  – Most common types of mandatory models work in multilevel security (MLS) environment
    • Typical in a military settings
    • Each subject and each object are assigned a security label
13.3 Architectural Access Control

13.3.2 Connector-Centric Architectural Access Control [1/13]

• An architectural description can be extended to model security and can be checked to examine whether the architecture successfully addresses the security needs of the system

• Basic concepts
  – The core concepts that are necessary to model access control at the architecture level: Subject, principle, resource, privilege, safeguard and policy
13.3 Architectural Access Control

13.3.2 Connector-Centric Architectural Access Control [2/13]

- **Subject**
  - A *subject* is the user on whose behalf a piece of software executes
  - Typically missing from software architectural models
  - Many software architectures assume that
    - All of its components and connectors execute under same subject
    - The subject can be determined at design-time
    - The subject generally will not change during run time, either inadvertently or intentionally, and
    - Even if there is change, it will have no impact on the software architecture
13.3 Architectural Access Control
13.3.2 Connector-Centric Architectural Access Control [3/13]

• **Principal**
  – A subject can take upon multiple *principles*
  – Principals encapsulate the credentials that a subject possesses to acquire permissions
  – Different types of credentials
  – Example: in a role-based access control model: each principle can denote the role that the user adopts

• **Resource**
  – A *Resource* is an entity for which access should be protected
  – Traditionally, resources are *passive* and accessed by active software components operating for different subjects
  – However, in software architecture, resources (e.g., components) can also be *active*
13.3 Architectural Access Control

13.3.2 Connector-Centric Architectural Access Control [4/13]

• *Permission, Privilege, and Safeguard*
  – *Permissions* describe operations on a resource that a component may perform
  – A *Privilege* describes what permissions a component possesses depending upon the executing subject
    • Privilege is important security concept that is missing from traditional architecture description languages
    • Two types of privileges corresponding to the two types of resources: (1) handles passive resources (2) handle active resources
  – A notion corresponding to privilege is *safeguard*: Describes conditions that are required to access the interfaces or protected components and connecters
13.3 Architectural Access Control
13.3.2 Connector-Centric Architectural Access Control [5/13]

- *Policy* specifies what privileges a subject, with a given set of principles, should have in order to access resources that are protected by safeguards

- Components and connecters consult the policy to decide whether an architectural access should be granted or denied
13.3 Architectural Access Control

13.3.2 Connector-Centric Architectural Access Control [6/13]

• The Central Role of Architectural Connecters
  – Architectural access controls is centered on connectors
  – Regulates communication between components
  – Supports secure message routing
13.3 Architectural Access Control

13.3.2 Connector-Centric Architectural Access Control [7/13]

- **Components: Supply Security Contract**
  - A security *contract* specifies the privileges and safeguards of an architectural element

- **Connecters: Regulate and Enforce Contract**
  - Play key role in regulating and enforcing the security contract specified by components
  - Determine whether components have sufficient privileges to communicate through the connecters
  - Provide secure interaction between insecure components
13.4 Trust Management

• Concerns how entities establish and maintain trust relationship with each other

• Plays a significant role in decentralized application
  – Entities do not have complete information about each other
  – Must make local decisions autonomously
  – Critical to choose an appropriate trust management scheme for a decentralized application
13.4 Trust Management

13.4.1 Trust

- A definition of trust coined by Diego Gambetta is
  - ... a particular level of the subjective probability with which an agent assesses that another agent or group of agents will perform a particular action, both before he can monitor such action (or independently of his capacity ever to be able to monitor it) and in a context in which it affects his own action.
  - The definition notes that trust is subjective and depends on view of the individuals; the perception of trustworthiness may vary from person to person
13.4 Trust Management

13.4.2 Trust Model [1/2]

- *Definition*: A **trust model** describes the trust information that is used to establish trust relationships, how that trust information is obtained, how that trust information is combined to determine trustworthiness, and how that trust information is modified in response to personal and reported experiences.
13.4 Trust Management
13.4.2 Trust Model [2/2]

• The trust model definition identifies three important components of a trust model
  – The first component specifies the nature of trust information used and the protocol used to gather that information
  – The second component dictates how the gathered information is analyzed to compute a trust value
  – The third component determines not only how an entity’s experiences can be communicated to other entities but also how it can be incorporated back into the trust model
13.4 Trust Management

13.4.3 Reputation-Based Systems [1/2]

• Related to trust is the concept of reputation
• Alfarez Abdul-Rehman and Stephen Hailes (Abdul-Rehman and Hailes 2000) define reputation as an expectation about an individual’s behavior based on information about or observations of its past behavior
• Trust management systems that use reputation to determine the trustworthiness of an entity are termed reputation-based systems
  – Example: Amazon.com and eBay
• Reputation-based systems can be either centralized or decentralized
13.4 Trust Management

13.4.3 Reputation-Based Systems [2/2]

- eBay
  - eBay is an electronic marketplace where diverse users sell and buy goods
  - eBay allows this trust information to be viewed through feedback profiles

- XREP
  - XREP is a trust model for decentralized peer-to-peer (P2P) file-sharing applications
  - No single point of failure
  - Increased robustness in addition to allowing users at the edge of the network
Managing Reputation and Trust in Peer-to-Peer Networks

- A peer–to-peer network
  - A decentralized network
  - All nodes in the network act as both clients and servers
  - Powered by the bandwidth of all peers
  - Notorious for the lack of reliability

- Types of P2P network
  - File sharing: Gnutella, Kazaa, BitTorrent
  - Instant messaging

- Existing trust and reputation systems
  - eBay, XREP
Trust Models: Common Terms

- Trust: A peer’s trust in other peers based on own past experience
- Reputation: A peer’s trust in another peer based on the experiences of other peers
- File provider: a peer providing a file to share
- Servent: A peer who is both client and server
- Free rider: A peer who only downloads and does not share any files
- Inauthentic files: Harmful files (viruses, corrupt, wrong file type; content not what it claimed to be)
P2P Security Concerns

• **Decoy files**: A malicious peer will respond to a download request, but will deliver a file that has malicious contents.

• **Malicious peer**: A peer who either belongs to one of malicious groups or will provide an inauthentic file for every request.

• **Malicious collective**: A group of malicious peers who know each other and collaborate to subvert a P2P system.

• **Self replication**: A virus such as Gnutella vbs_worm poses as a peer and then creates a copy of itself for download.

• **Pseudospoofing**: Malicious peers control multiple identities; false pseudonyms are used to give good reputation to other pseudonyms controlled by the same malicious peer.
P2P Trust and Reputation

• Peers store opinions on their experiences in downloading resources (files)
• Peers store an opinion about the resource provider and the resource (file)
• The opinions are computed into binary or another mathematical probability
• Peers share their opinions, providing recommendations for resource providers and resources
• The aim of the system is to eliminate malicious peers and inauthentic files
XREP Reputation Model [1/2]

• XREP: A reputation based approach for choosing reliable resources in P2P networks
• A distributed polling algorithm allows reputations to be shared
• Each peer is termed a “servant” because of its ability to be a server and client
• Each peer maintains peers reputations in the system
• Each peer also evaluates resources accessed from peers
XREP Reputation Model [2/2]

• Each servant maintains its own experience on resources and other peers and shares with other servants upon request
  – Resource repository: records an ID for each file downloaded and whether it is good (+/1) or bad (-/0)
  – Servent repository: each servant stores the number of successful and unsuccessful downloads for each servant
XREP Distributed Algorithm

• XREP: a distributed protocol that allows reputation values to be maintained and shared

• The algorithm has five phases
  1. Resource searching
  2. Resource selection and vote polling
  3. Vote evaluation
  4. Best servant check
  5. Resource downloading
XREP Phase 1: Resource Searching

- Similar to Gnutella (A GNU project, 2000)
- A servent, p, broadcasts to all its neighbors a Query message containing search keywords
- When a servent, s, receives a Query, it responds with a QueryHit message
XREP Phase 2: Resource Selection and Vote Polling

• Once QueryHits are received, the originator (p) selects the best matching resource

• The originator (p) polls other peers using a Poll message to inquire their opinion about the resource and the servent

• Each peer responds (provides a vote) with a PollReply
XREP Phase 3: Vote Evaluation

• The originator (p) collects all votes and verifies (in detail) the authenticity of the PollReply messages
  – Cluster computation
  – Guard against a group of malicious peers acting in tandem
  – Sends TrustVote to peers and requests confirmation
XREP Phase 4: Best Servant Check

- Several servants may have offered the resource
- The originator (p) must choose a reliable servant
- The servant with best reputation is chosen
- Checks for availability
XREP Phase 5: Resource Downloading

• The resource is downloaded from the servent that is available

• The originator (p) will update its repositories with its opinion on the downloaded resource and the servent
13.4 Trust Management

13.4.4 Architectural Approach to Decentralized Trust Management [1/27]

• Decentralized systems and their susceptibility to various types of attacks makes it critical to carefully design such systems

• Software architecture provides an excellent basis to reason about these trust properties

• Three main steps involved in an architectural approach
  – Understanding and assessing the real threats to a system
  – Designing countermeasures against these threats
  – Incorporating guidelines corresponding to these countermeasures into an architectural style
13.4 Trust Management

13.4.4 Architectural Approach to Decentralized Trust Management [2/27]

• Threats to Decentralized Systems
  – *Impersonation*
    • Malicious peers may attempt to conceal their identities by portraying themselves as other users
    • Targets of a deception need the ability to detect these incidents
  – *Fraudulent Actions*
    • Malicious peers may act in bad faith without misrepresenting themselves (claim a service without actually having
    • The system should attempt to minimize the effects of bad faith
13.4 Trust Management

13.4.4 Architectural Approach to Decentralized Trust Management [3/27]

• Threats to Decentralized Systems [continued]
  
  – *Misrepresentation*: Malicious users may misrepresent their trust relationships with other peers in order to confuse other peers

  • Peers could publish that they do not trust an individual that they know is trustworthy

  • Peers could claim that they trust user that they know to be dishonest
13.4 Trust Management
13.4.4 Architectural Approach to Decentralized Trust Management [4/27]

• Threats to Decentralized Systems [continued]
  – Collusion (secret agreement)
    • A group of malicious users may also join together to actively subvert the system
    • A certain level of resistance needs to be in place to limit the effect of malicious collectives
13.4 Trust Management

13.4.4 Architectural Approach to Decentralized Trust Management [5/27]

• Threats to Decentralized Systems [continued]
  – Denial of service
    • In an open architecture, malicious peers launch an attack on individual or groups of peers
    • These attacks may flood peers with well-formed and ill-formed messages
    • In order to compensate, the system requires an ability to contain the effects of denial of service attacks
13.4 Trust Management

13.4.4 Architectural Approach to Decentralized Trust Management [6/27]

- Threats to Decentralized Systems [continued]
  - *Addition of Unknowns*
    - The cold start situation: upon initialization, a peer does not know anything about anyone else on the system
    - The ability to bootstrap relationships when no prior relationships exist is essential
13.4 Trust Management

13.4.4 Architectural Approach to Decentralized Trust Management [7/27]

• Threats to Decentralized Systems [continued]
  – *Deciding Whom to Trust*: when gradually observing good and dishonest behavior
    • Trust relationships generally should improve when good behavior is perceived of a particular peer
    • Trust relationship should also generally be downgraded when dishonest behavior is perceived
13.4 Trust Management

13.4.4 Architectural Approach to Decentralized Trust Management [8/27]

• Threats to Decentralized Systems [continued]
  – *Out-of-Band Knowledge*
    • Out-of-band knowledge occurs when there is data not communicated though normal channels
    • Trust is based on visible interactions
    • There may also be invisible interactions
    • Therefore, ensuring the consideration of out-of-band information is essential (A trusts B, B trust Cs, thus A should trust C)
13.4 Trust Management

13.4.4 Architectural Approach to Decentralized Trust Management [9/27]

• Measures to Address Threats
  – *Use of Authentication*
    • To prevent impersonation attacks, it is essential to use some form of authentication so that message senders can be uniquely identified
    • Example: signature-based authentication
13.4 Trust Management

13.4.4 Architectural Approach to Decentralized Trust Management [10/27]

• Measures to Address Threats [continued]
  – Separation of Internal Beliefs and Externally Reported Information
    • A peer may have its own individual goal which may conflict with those of other entities
      – favor a particular peer
      – trust own data
      – don’t disclose sensitive data
    • Therefore, it is important to model externally reported information separately from internal beliefs
13.4 Trust Management
13.4.4 Architectural Approach to Decentralized Trust Management [11/27]

• Measures to Address Threats [continued]
  – *Making Trust Relationships Explicit*
    • Without a controlling authority that governs the trust process, peers require information from other peers to make decisions
    • Active collaboration may be essential
    • It is important for the trust relationships to be explicit and exchangeable between peers
13.4 Trust Management

13.4.4 Architectural Approach to Decentralized Trust Management [12/27]

• Measures to Address Threats [continued]
  – *Comparable Trust*
    • Ideally, published trust values should be syntactically and semantically comparable
    • If the same value has different meaning across implementation, then accurate comparisons across peers cannot be made
13.4 Trust Management

13.4.4 Architectural Approach to Decentralized Trust Management [13/27]

• Corresponding Guidelines to Incorporate into an Architectural Style
  – *Digital Identities*
    • Without the ability to associate identity with published information, it is a challenge to develop meaningful relationships
    • Important to understand the limitations of digital identities with respect to physical identities
    • Architectural style should consider trust relationships only between digital identities
13.4 Trust Management

13.4.4 Architectural Approach to Decentralized Trust Management [14/27]

• Corresponding Guidelines to Incorporate into an Architectural Style [continued]
  
  – *Separation of Internal and External Data*
    • Explicit separation of internal and external data supports the separation of internal beliefs from externally reported information within a peer
    • The architectural style should adopt the explicit separation of internal and external data
13.4 Trust Management

13.4.4 Architectural Approach to Decentralized Trust Management [15/27]

- Corresponding Guidelines to Incorporate into an Architectural Style [continued]
  - Making Trust Visible
    - Trust information received externally from entities is used within the peer architecture to make local decisions
    - Therefore, the architecture style should require trust relationship to be visible to components in the peer’s architecture as well as be published externally to other peers
13.4 Trust Management

13.4.4 Architectural Approach to Decentralized Trust Management [16/27]

• Corresponding Guidelines to Incorporate an Architectural Style [continued]
  – Expression of Trust
    • There has been no clear consensus in the trust literature as to which trust semantics provide the best fit for application
    • No need to indiscriminately enforcing a particular trust semantic
    • Only request that trust values be syntactically comparable
13.4 Trust Management

13.4.4 Architectural Approach to Decentralized Trust Management [17/27]

• Resultant Architectural Style
  – The above principles and constraints can be combined to create an architectural style for decentralized trust management
  – In addition four functional units of a decentralized entity are identified:
    • Communication: handles *interaction with other entities*
    • Information: responsible for persistently *storing trust* and application-specific information
    • Trust: responsible for *computing trust worthiness* and guides trust-related decisions
    • Application: includes application-specific functionality and is responsible for enabling *local decision making*
13.4 Trust Management
13.4.4 Architectural Approach to Decentralized Trust Management [18/27]

- Resultant Architectural Style [continued]
  - Given the interplay between the four units, adopting a layered architectural style offers reusability of components
  - Example: C2
  - PACE extends C2
    - Practical Architectural Style for Composing Egocentric applications
    - Includes all of the above four units
13.4 Trust Management

13.4.4 Architectural Approach to Decentralized Trust Management [19/27]
13.4 Trust Management

13.4.4 Architectural Approach to Decentralized Trust Management [20/27]

- **PACE-Induced Benefits**
  - *Impersonation*: Impersonation refers to the threat caused by a malicious peer posing as another in order to misuse the peer’s privileges or reputation
    - Addresses this threat through use of digital signatures and message authentication
    - All external communications constrained to the Communication unit
    - PACE components refuse to assign trust values to revoked public keys
13.4 Trust Management

13.4.4 Architectural Approach to Decentralized Trust Management [21/27]

- PACE-Induced Benefits [continued]

  - *Fraudulent Actions:* Malicious peers may engage in fraudulent behavior including advertising false resources or services and not fulfilling commitments

    - Since PACE is designed for open, decentralized architectures there is little that may be done to prevent the entry of malicious peers
    - Malicious actions may be detected by the user or through the Application layer
13.4 Trust Management
13.4.4 Architectural Approach to Decentralized Trust Management [22/27]

• PACE-Induced Benefits [continued]
  – *Misrepresenting Trust*: A malicious peer may misrepresent its trust with another in order to positively or negatively influence opinion of a specific peer
    • Since PACE facilitates explicit communication of comparable trust values, a peer can incorporate trust relationships of others
    • A transitive trust model: If A trusts B and B trusts C, then A can trust C
13.4 Trust Management

13.4.4 Architectural Approach to Decentralized Trust Management [23/27]

• PACE-Induced Benefits [continued]
  – Collusion: refers to the threat by a group of malicious peers
  • It has been proven that explicitly signed messages between peers can overcome a malicious collective in a distributed system
13.4 Trust Management

13.4.4 Architectural Approach to Decentralized Trust Management [24/27]

• PACE-Induced Benefits [continued]
  
  – Denial of Service: Malicious peers may also launch attacks against peers by flooding them with well-formed or ill-formed messages
  
  • The separation of the communication layer allows isolation and response to the effects of such denial of service attacks
  
  • For well-formed message, the Communication Manager can limit message transmission rates
13.4 Trust Management

13.4.4 Architectural Approach to Decentralized Trust Management [25/27]

• PACE-Induced Benefits [continued]

  – *Addition of Unknowns*: when the system is first initialized, there can be a cold-start problem because there are no existing trust relationships

  • PACE’s application layer can still receive messages
  • Without enough information to make an evaluation, the Trust Manager will not assign a trust value to the message
13.4 Trust Management

13.4.4 Architectural Approach to Decentralized Trust Management [26/27]

• PACE-Induced Benefits [continued]
  
  – *Deciding Whom To Trust*: In a large-scale system, certain domain-specific behaviors many indicate the user’s trustworthiness

  • In PACE, the application trust rules component allows for automated identification of application-dependent patterns
13.4 Trust Management
13.4.4 Architectural Approach to Decentralized Trust Management [27/27]

• PACE-Induced Benefits [continued]
  – Out-of-Band Knowledge
    • It is essential to ensure that out-of-band information is also considered in establishing trust relationships
    • While PACE confines all electronic communication to the communication layer, out-of-band trust information can originate as requests from the user through an application layer
13.5 End Matter [1/2]

• Presented principles, concepts and techniques relevant to the design and construction of secure software systems

• The emphasis has been on showing how security concerns may be assessed and addressed as part of a system’s architecture

• Connecters can be used to examine all message traffic and help ensure that no unauthorized information exchange occurs
13.5 End Matter [2/2]

• Explored how the use of a particular architectural style, attuned to security needs in open decentralized applications, can help mitigate risks

• Trust models are central to this approach

• Making system secure may, however, compromise other non-functional requirements