Security Risks in Systems of Distributed Objects, Components, and Services

David Chizmadia

Promia, Inc

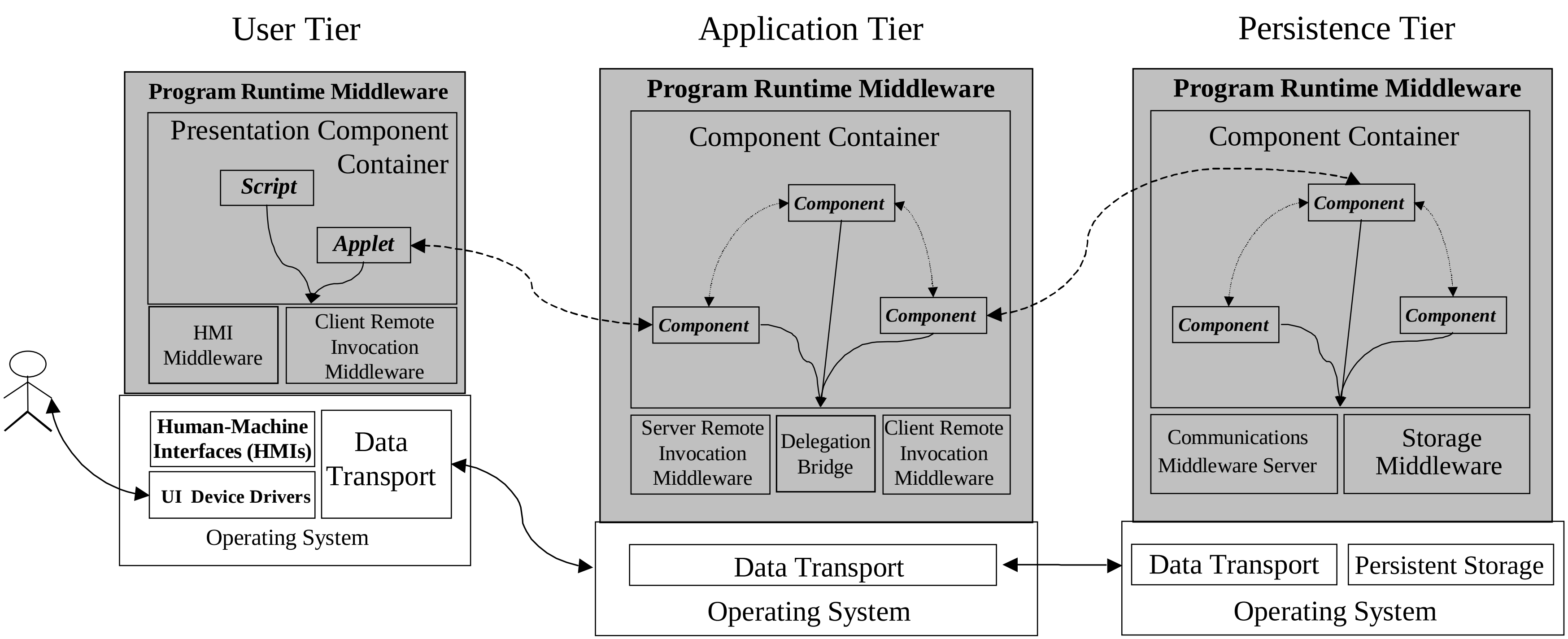
dchizmadia @ promia.com

(410) 694-0322

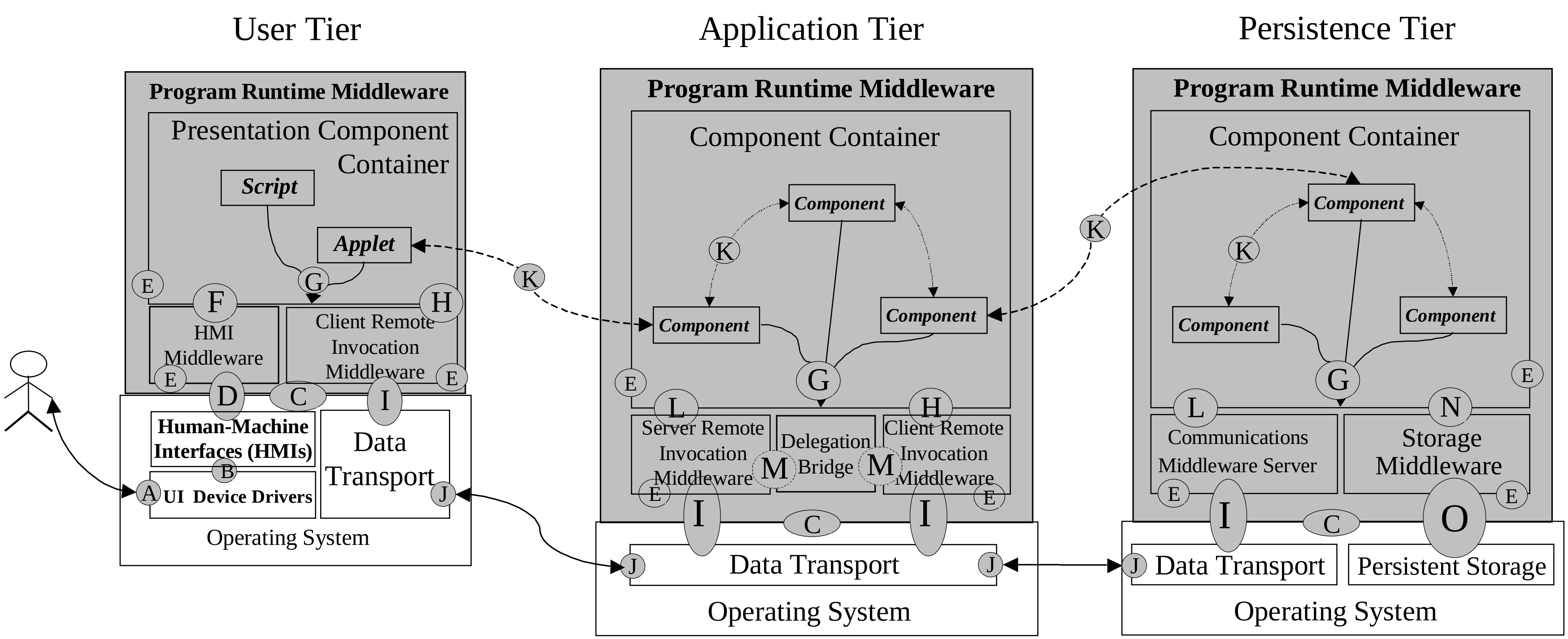
Objectives

- Provide a “Big-Picture” perspective on the full range of security risks that are present in **ANY** system comprised of distributed, communicating components
- Motivate the specific discussions of DOCSec topics for the rest of the workshop

Elements In Systems Of Distributed Objects, Components, And Services



Security Relevant Boundaries In Systems Of Distributed Objects, Components, And Services



O => **15** different security boundaries in DOCS systems

User

- User is the “prime cause” of most application activity
- From a security risk perspective, Users are
  - Unpredictable
  - Creatively error-prone
  - Potentially inimical to application security policy
- User-tier usually consists of:
  - commodity hardware (Intel, Mac, Palm)
  - commodity OS (Windows, MacOS, Linux, PalmOS)
  - a web browser
- User-tier is often in an uncontrolled environment with minimal physical security

Security Risks: User vs User-Tier

- Unauthorized user impersonating authorized user
  - Special case: authorized user leaves secure user-tier session unattended
- Authorized user exceeds assigned authorization
- Malicious software on user-tier computer impersonates user-tier security component(s)

Node Operating System

- Provides basic resource management for any tier
- Always includes process-related functions, including
  - Process creation, scheduling, and destruction
  - Loading/Unloading code from a process
  - Inter-process communication (IPC)
  - Security and other process metadata management
- May also include functions for:
  - User interface input and output device drivers
  - Human-Machine Interfaces (e.g., windowing or voice recognition)
  - Data transport protocol software
  - Persistent storage services (e.g., files or databases)

Security Risks: Node OS vs Processes

- Malicious code gains control of a process
  - Attempts to over-consume resources such as CPU cycles, main memory, or disk space
  - Attempts to interfere with Node OS operation
  - Attempts to interfere with operation of other processes

User Interface Device Drivers

- User devices provide physical communication path between user and user-tier hardware
- UI Device Drivers translate physical events into software events
- In many user-tier Node OS, device drivers run in the same address space and with the same authorities as the OS

Security Risks: Device Drivers vs Node OS

- Malicious device drivers have same authority as Node OS
  - Modify OS security data structures
  - Modify scheduling of other processes
  - Modify security metadata of user processes

Security Risks: Device Drivers vs Users

- Compromised device drivers can log user security interactions (e.g., entering password)
- Compromised device drivers can allow a malicious process to take control of user security interactions
- Compromised device drivers can modulate device circuitry to transmit data

Security Risks: Device Drivers vs HMIs

- Malicious device drivers can masquerade as users
  - Scripted user security interactions
  - Enable Man-In-The-Middle attacks

Program Runtime Middleware

- Provides a common environment within which components can execute.
- Simplest for of PRM is a portable system library - i.e., libc
- PRM could also include a virtual machine specification and a common set of libraries that allow code to run unchanged on multiple hardware and software platforms.
- The purpose of the PRM is to define a common, virtual, semantics for the OS to which portable application components can be written, irrespective of the OS actually being used

Security Risks: PRM vs Node OS

- Subtle differences between the PRM and the semantics of a specific OS lead to
  - incorrect PRM use of the OS functions
  - Incorrect selection and use of the security operating parameters for a specific OS
- Differences between the resources protected by the PRM and the OS
  - OS protected resources are a subset of PRM protected resources
  - PRM protected resources are constructed out of OS protected resources

Security Risks: PRM vs PRM Clients

- PRM clients make use of both PRM and OS protection
- PRM clients exploit semantic mismatches between PRM and OS security models
- Malicious PRM colludes with malicious PRM Client to disclose information in other PRM Clients
- Malicious PRM colludes with malicious PRM Client to modify or destroy information in other PRM Clients
- Malicious PRM interferes with execution of PRM Clients

Human-Machine Interface Middleware

- Implements a set of interfaces by which any program can interact with a virtualized set of HMIs
- Security relevance is in fact that User I&A and resource naming and protection instructions must pass through the HMI Middleware

Security Risks: HMI Middleware vs HMIs

- In very complex HMI Middleware, many virtual HMI objects will be constructed out of primitive platform HMI objects
  - Potential for platform-specific protection mismatches
  - Additional potentil for incorrect HIM Middleware object implementation, which can affect all platforms

Security Risks: HMI Middleware vs Container

- Malicious HMI Middleware can masquerade as users
  - Scripted user security interactions
  - Enable Man-In-The-Middle attacks
  - Gains leverage as a multi-platform risk

Component Container

- Containers primarily hold software components
  - term is used here to also refer to browser ability to run applets
- Container provides the services needed by components to interact with external world
  - events/notification, transactions, persistence, and security
- Container manages component lifecycle and execution state
- In most current systems a Container runs as a single process under a single identity

Security Risks: Container vs CO

- Compromised Container can modify CO’s code base
- Container can disregard or not respond to CO’s security requests

Security Risks: Container vs HMI Middleware

- Container can corrupt data flowing to or from HMI Middleway
- Compromised Container can pass improperly received data to compromised HMI Middleware for covert external transmission
- Compromised Container can interpose itself in a security dialogue between a User and a Contained Object providing a security service

Security Risks: Container vs Client RIM

- A compromised Container can misrepresent its identity or authority to the Client RIM
- Container could attempt to repudiate a Request that it sent
- Container could inappropriately disclose local security metadata in a Request

Security Risks: Container vs Server RIM

- A compromised Container can misrepresent its identity or authority to the Server RIM
- Container could attempt to repudiate a Reply that it sent
- Container could inappropriately disclose local security metadata in a Reply

Security Risks: Container vs Storage Middleware

- Container could fail to indicate CO instance state that needs protection while in storage
- Saved component instance state could be accessed by unauthorized users
- Saved component instance state could be modified by unauthorized users

Contained Object (CO)

- Goal of Container design pattern is to allow creation of COs with common, well-defined functions that can be combined to meet specific application requirements
- COs are assumed to present and use well-defined interfaces to and from other COs and the Container
- Reminder from Container: In many Container implementations all COs run with all of the authority of the Container

Security Risks: CO vs Container

- Malicious CO could discover and misuse ambient Container authority
- Malicious CO could attempt to compromise Container's security configuration
- Malicious CO could attempt to compromise Container code base
- Malicious CO could attempt to interfere with the operation of the Container and its COs

Security Risks: CO vs Other Local CO

- Components may attempt to modify each other
- Security architecture may warrant components with independent identity and authorization
- Malicious component could masquerade (e.g., present same inter-component interfaces) as another component

Security Risks: CO vs Remote CO

- Security risks below apply to either direction of interaction
  - Malicious CO could masquerade (e.g., present same inter-component interfaces) as another CO
  - Malicious CO could intentionally send data for which the other CO is not authorized
  - Uncompromised CO could receive data for which it isn’t authorized
    - And notice, or
    - Not notice and continue processing
  - Malicious CO could attempt to repudiate an exchange

Client-Side Remote Invocation Middleware

- Invocations transferred by remote invocation middleware (CORBA, Web Services, RMI)
- Target of invocation (server) is identified by a reference to hide details of server location and implementation from client
- Remote invocation middleware handles resolution of server location and translation of data when client and server use any combination of different hardware, OS, and programming language

Security Risks: Client RIM vs Container

- Spoofing of the Server Reference
- Fail to provide protection to Security-Unaware Containers
- Inappropriate protection of request integrity and confidentiality
- Client Authentication Domain Different From Server Authentication Domain
- Client Authorization Domain Different From Server Authorization Domain

Security Risks: Client RIM vs Data Transport

- Unwanted disclosure of Client host’s existence
- Incorrect use of data transport security mechanisms

Security Risks: Client RIM vs Server RIM

- Different interpretation or implementation of security metadata protocol
  - Causing failure to interoperate securely
- Client Authentication Domain Different From Server Authentication Domain
- Client Authorization Domain Different From Server Authorization Domain

Data Transport

- Communication middleware still relies on standard data transport abstractions to transfer data - it just hides details from components
- Most kinds of communications middleware are evolving to allow invocations to occure over a variety of data transports
  - e.g., TCP/IP, HTTP, SCTP, etc

Security Risks: Data Transport vs Client RIM

- Misrouting or dropping Requests
- Spoofing of the Server Identity and other security metadata
- Inappropriate Protection of Request Integrity and Confidentiality

Security Risks: Local vs Remote Data Transport

- Essentially threats arising from network fabric
  - Incorrect implementation of data transfer protocols
  - Passive eavesdropping leading to disclosure of request contents
  - Active eavesdropping leading to modification or destruction of request contents
  - Inability to cross network boundaries (e.g., firewalls)
  - Flooding attacks against specific nodes or subnets

Security Risks: Data Transport vs Server RIM

- Misrouting or dropping Responses
- Spoofing of the Client Identity and other security metadata
- Inappropriate Protection of Response Integrity and Confidentiality

Server-Side Remote Invocation Middleware

- Accepts client invocation requests
- Resolves target of request to a component within a container
- Converts request into a language-specific call up to the component
- Converts response from component into an invocation reply that is returned to the client

Security Risks: Server RIM vs Data Transport

- Unwanted Revelation of Server Host Existence
- Incorrect use of data transport security mechanisms

Security Risks: Server RIM vs Container

- Server RIM could forward undesired or inappropriate invocations to Container or CO
- Spoofing of the Client Identity
- Inappropriate Protection of Response Integrity and Confidentiality
- Flooding Attacks Against Specific Components, Containers, or Servers

Security Risks: Server RIM vs Client RIM

- Different interpretation or implementation of security metadata protocol
  - Causing failure to interoperate securely
- Placing incorrect information in Server Reference
- Server Authentication Domain Different From Client Authentication Domain
- Server Authorization Domain Different From Client Authorization Domain

Delegation Bridge

- In DOC systems, the component a client invokes may invoke other components in other containers, possibly on other server hosts, to process the request
- Some security policies may require a client to allow the other components to be invoked with some, or all, of the originating client's authorizations
  - i.e., the client must "delegate" authorization to the component
- In emerging environment, the Server Middleware may be a different technology than the Client Middleware
  - e.g., Server may accept Web Services invocations, while Client may make CORBA invocations

Security Risks: Same RIM Technology

- Concept of delegation is misunderstood
- The intermediate delegation model is incompatable with the security policy of the originating client
- The intermediate delegation model is incompatable with the security policy of one or more downstream components
- The originating client, intermediate component, and target component are in different Authentication Domains
- The originating client, intermediate component, and target component are in different Authorization Domains
- The security information received by an intermediate component is incomplete or incompatable with the security information required to invoke a downstream component
- Transitive delegation cannot be enforced
- An authorized intermediate component is compromised and intentionally delegates incorrectly

Security Risks: Different RIM Technologies

- Mismatch in encoding syntax may lead to loss or corruption of security information
- Security metadata may be lost if one RIM technology security model is less expressive than the other
- All of the same-RIM-technology security risks

Storage Middleware

- Interface between the Container and an external storage system - usually a database
- Makes it appear that a component instance has been saved
- Ultimately only saves the data from the component instance

Security Risks: Storage Middleware vs Container

- Failure to store component state
- Mismatch between storage middleware security policy(s) (if any) and Container security policy(s)

Security Risks: Storage Middleware vs PSS

- Saved component instance state could be accessed by unauthorized users
- Saved component instance state could be modified by unauthorized users

Persistent Storage System

- Actually saves the data that storage middleware extracts from a component instance
- Is tightly bound to the storage mechanism (filesystem, database)
- Must interact with any security mechanisms found in the storage mechanism

Security Risks: PSS vs Storage Middleware

- Unexpected interaction between storage middleware and storage system security policy or mechanism
- Storage System, Persistence System, and Container/ Component are in Different Authentication Domains
- Storage System, Persistence System, and Container/ Component are in Different Authorization Domains

Security Risks: Node OS vs PSS

- Unauthorized access to component state by authorized storage system administrators
- Unauthorized access to component state by unauthorized users of the storage system

Pervasive Security Risks

System-wide potential risks that can apply to each or all of the parts that comprise the system

- Configuration Flaws
- Software Flaws
- Component Design Flaws
- Component Composition Flaws
- Application Design Flaws

Security Risks: Configuration Flaws

- Software (and/or hardware) is not installed and configured according to security guidance
- Can be very subtle
- Can lead to cascading security exposures in a DOCSec system

Security Risks: Software Flaws

- Each component will inevitably have coding errors
  - Buffer overflows
  - Incorrect or non-existent input or returned value validation
- Some errors won't be found by analysis and testing
- Errors in interfaces that deal with security introduce security risks to both:
  - The individual component, and
  - Other components that rely on the correctness of that component's security functions

Security Risks: Component Design Flaws

- The design for a component doesn’t correctly realize the security requirements identified for that component
- “Bad design drives out good”
- Intrinsic errors in the component interfaces that deal with security introduce security risks to both:
  - The individual component, and
  - Other components that rely on the correctness of that component’s security functions

Security Risks: Component Composition Flaws

- Essentially same problem as configuration flaws…
- Each component is correct with respect to its security requirements
- Components are improperly assembled into a “super-component”
- Unexpected “emergent” security properties from the collection of components

Security Risks: Application Design Flaws

- Incomplete, incorrect, or imprecise application security requirements
- Mismatch between the security features the application needs and ones the components provide
- The design of user-facing application components doesn’t adhere to the “principle of least surprise”

Questions?
Comments?