æSec Corporation

The power of verifiable protection™

Creating High Assurance for a product: Lessons Learned from GEMSOS

Roger Schell
Roger.Schell@aesec.com
(831) 657-0899

IEEE IWIA 2005
College Park, MD
March 23, 2005

Insecure Products & Secure Bricks Are Easy – many worked examples

- Building a secure anything is hard
  - Verifiable enforcement of information security policy

- Building a supportable product is hard
  - Research and Government development don't scale

TODAY TOPIC:

How to build a high assurance secure product

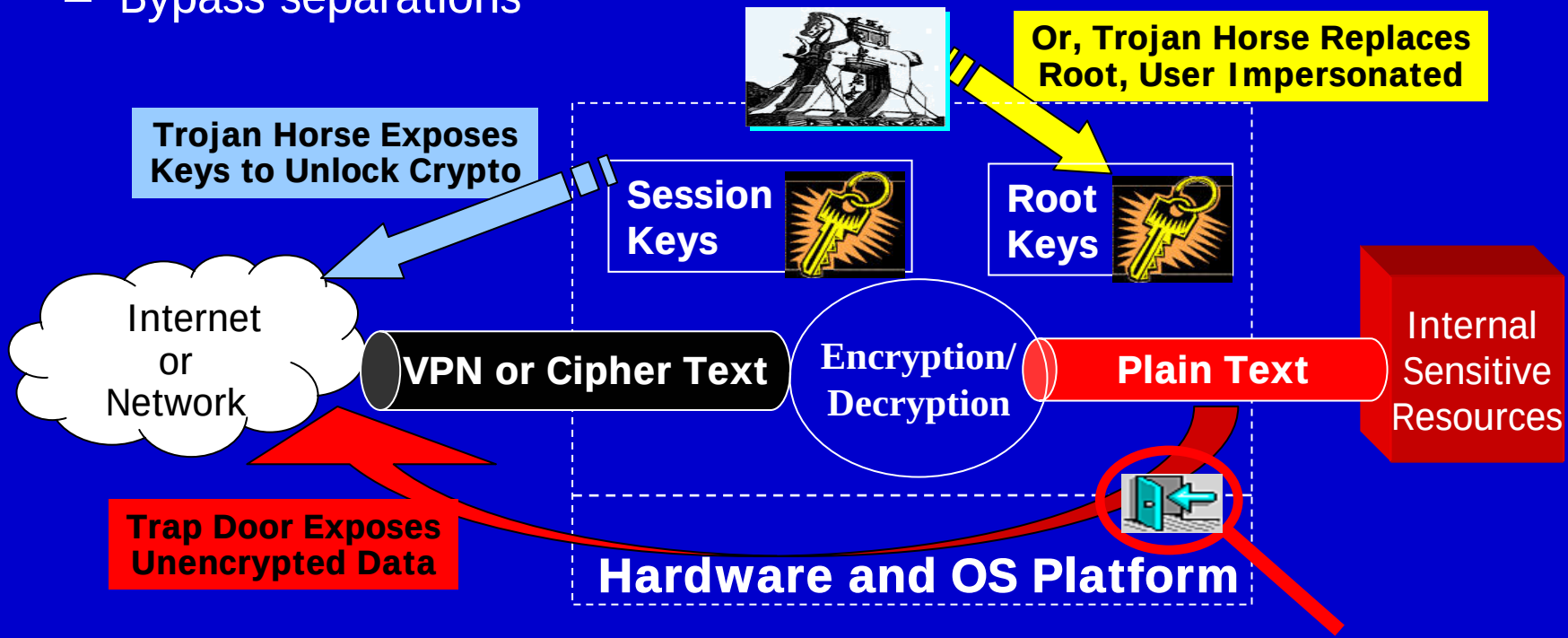
- Lessons learned from development of GEMSOS (Gemini Multiprocessing Secure Operating System)
- Deployed as secure, useful & supportable OEM product

Need for High Assurance: The Subversion Problem

- Historically called the Multilevel Security (MLS) problem
  - Prevents interconnection of networks
  - Impedes real-time sharing of information
- Trusting computer technology to prevent grave damage
  - Software of uncertain pedigree
  - Commercial software and “open source” easily subverted
  - Much overseas – effectively using software from our enemies
- Early policy limited interconnections and sharing
  - Policy defined “Dedicated Mode” and “Controlled Mode”
  - Big pressure for more connections, called “Multilevel Mode”
- Can you buy your solutions from your mortal enemy?
  - (Better figure out how to, because in reality you likely are)
- “Tiger Teams” show subversion is attacker tool of choice
  - http://www.airpower.maxwell.af.mil/airchronicles/aureview/1979/jan-feb/schell.html
  - http://www.acsac.org/2002/papers/classic-multics.pdf

Need for High Assurance: Malicious Software

- **Professionals use trap doors, Trojan Horses, worms, etc.**
  - Steal keys, forge certificates, or steal plain text
  - No need to crack cryptography
  - Bypass separations



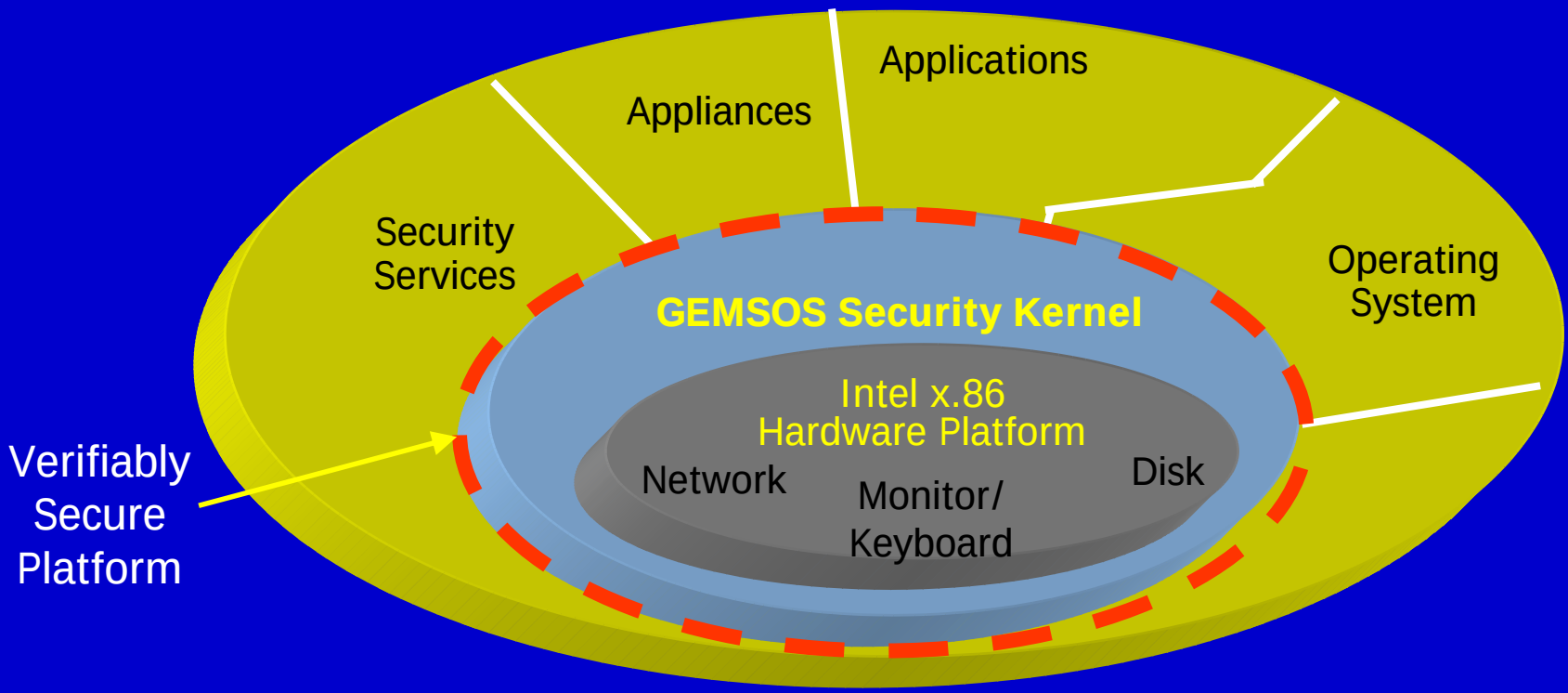
- **Easy to subvert Trusted Solaris, SE Linux, Windows XP, IPSEC . . .**
  - Examples: "Subversion as a Threat in Information Warfare", Journal article (2004)

Need for High Assurance: Infrastructure Subversion

- "Trap Doors"
  - Malicious Flaws in Foundation
  - Software, e.g., Operating System or Tools
  - Hardware, e.g., BIOS in PROM Bypasses Protection
  - Efficacy and Effectiveness Demonstrated
- Activated by unique "key" or trigger
  - Known only to attacker
  - Bootstrap for diverse attacks
- Planned, Deliberate Insertion
  - Supply Separate Software Module
  - "Plant" Development Staff Member
  - Exploitable by Malicious Applications
  - High Potential Future Benefit to Adversary
  - Testing Not at all a Practical Way to Detect

What Was Built

- General purpose multiprocessing operating system kernel
  - Application-agnostic
  - Good performance and multiprocessing
- Demonstrable real-world policy enforcement
  - Controlled sharing of information per customer policy
  - Independently evaluated – buy it from your mortal enemy

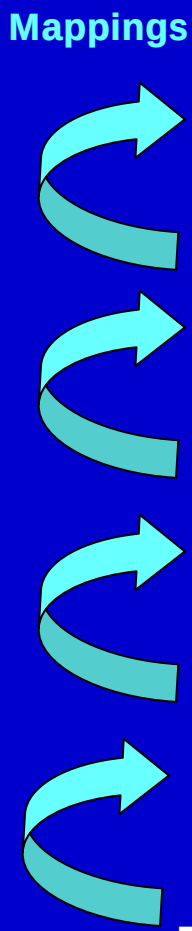


Creating High Assurance (Order of TCSEC Class A1/EAL7)

- Solutions must be based in science
  - Policy statement about information & people
  - Formal security model of a reference monitor
  - Formal specification proven to map to model
  - Implementation that demonstrably corresponds to spec

- Getting 90% done is not that hard
  - The remaining 10% can become impossible
  - High assurance requires 100% *completeness*
  - “New paradigms” may not reach 90% until evaluation

Evaluation Evidence for Verifiable Protection



Mappings

Security Policy

Philosophy and Design of Protection

Formal Security Policy Model

Formal Top Level Specifications

Descriptive Top Level Specifications

Implementation Design Documents

Source Code

Covert Channel Analysis

Product End Item

Trusted Distribution

Security Features Users Guide

Trusted Facility Manuals

System Specifications

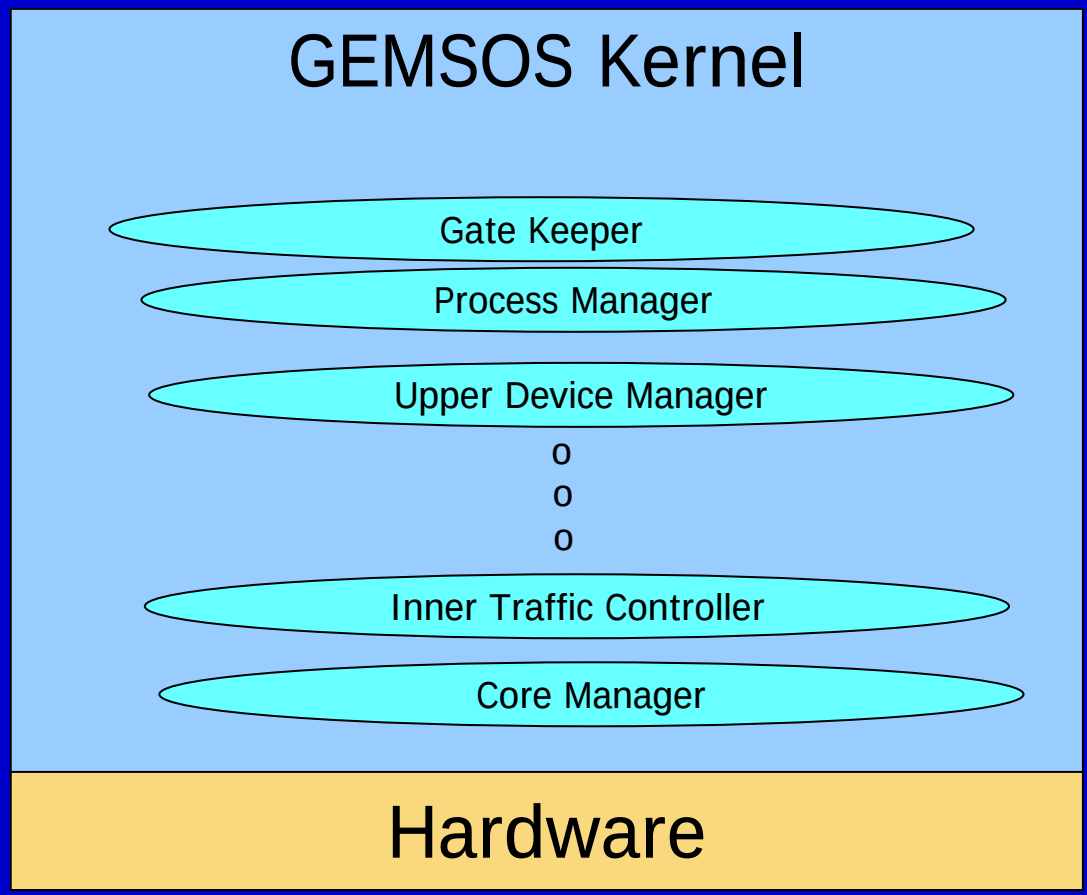
Development Specifications

Product Specifications

Deliverable Product

Loop-free Layers of Independent Modules

True layering means I can separately compile and test each layer in the complete absence of higher layer code. The next time you hear “high assurance” ask, how many layers and are they loop free?



Information Hiding Modules

- Historically referred to as “Parnas modules”
- Construction is an art – there are no cookbooks
  - Input parameters passed by value
  - Easy to end up with bad code
- Not just “object oriented” programming
- Effective use of hardware to isolate module data
- Modules form layers
  - Lower layers serve as lemmas for higher layers
  - Informal proof of implementation

Minimization

- Exclude non-security relevant functions
  - A challenge to rid programmers of function creep
- Virtualize the hardware; low level device drivers
  - Upside is most driver modifications don't touch kernel
- Support for excluded functions outside the kernel
  - Evaluated enforcement of protection domains helps
  - Protect non-kernel O/S from applications

Practical implication:

Kernel cannot be compatible with insecure O/S

Development Best Practices

- Strict avoidance of classic trouble spots (e.g., recursion, conditionally compiled code, untyped buffers)
- Strongly typed programming (use of Pascal)
- Single entry / single exit subroutines
- Parameter passing conventions – pass by value
- Restricted manipulation of pointers
- Separation of initialization and run-time code

Configuration Management

- All items through the entire life cycle
- A lot more than just source code control
  - From specifications through end item (object code)
- The foundation is the manual procedures
  - Not dependent on complex (subvertable) tools
  - Emphasis on being inspectable by humans
  - Ability to show deltas

Trusted Product Distribution

- How does customer confirm product is legit?
  - Not tampered with in route
  - Actually the evaluated version
- Evaluated public key cryptography-like solution
  - Unique cryptographic keys for each system
    - System-specific keys
    - Community keys
  - OEMs can establish distribution independent of vendor
    - Vendor knowledge of keys cannot lead to subsequent subversion
  - End customer can protect self from OEM & Vendor

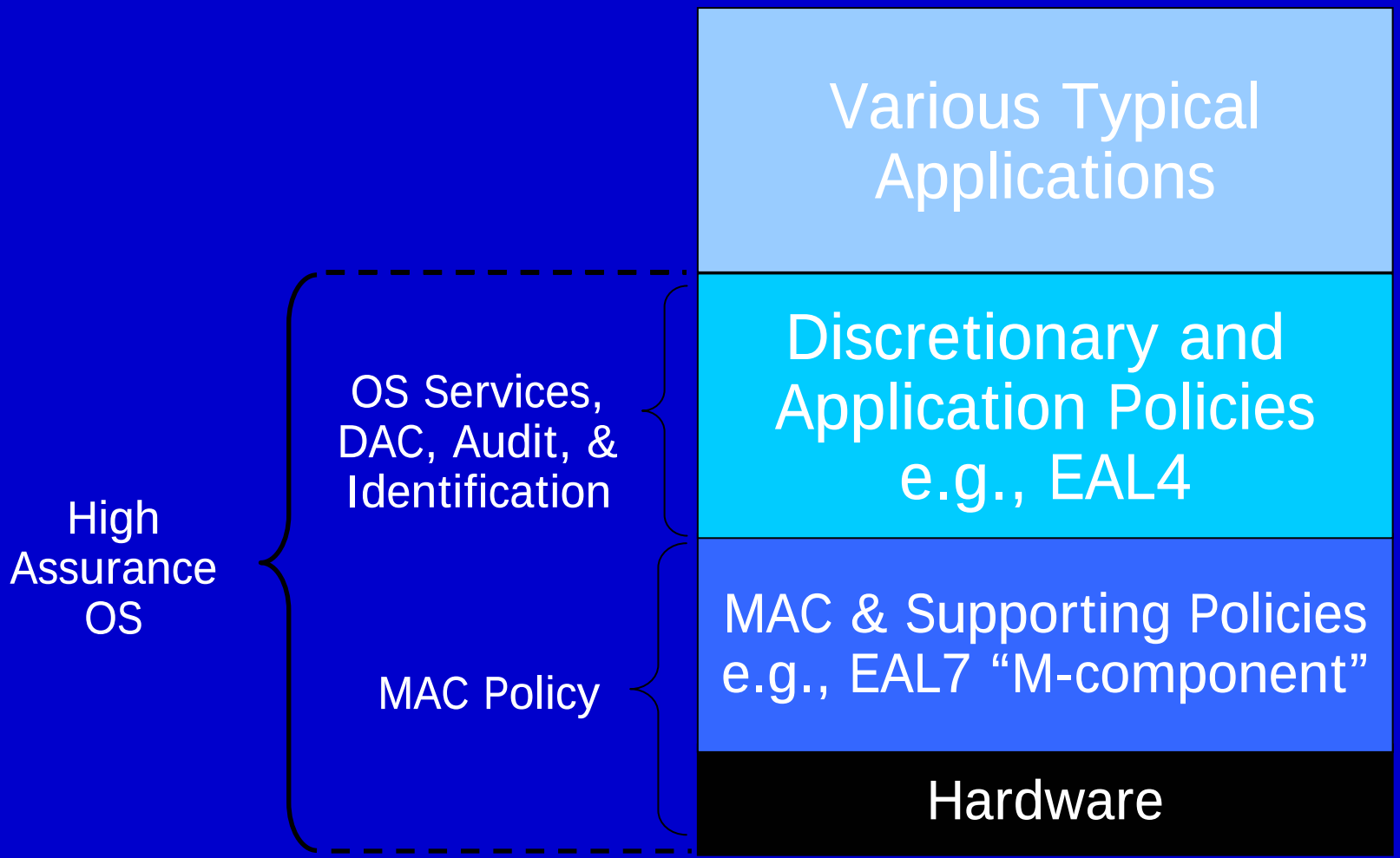
Solutions Based in Science

- Divide and Conquer
  - Not All Problems Are the Same
  - Fundamental Ability to Protect Information is Critical
- Distinguish Between Policies
  - Some Enforced With High Assurance, Some Can't Be
  - Some Violations Result in Massive Loss – Some Do Not
- Don't Have to Solve All Hard Problems at Once
  - Spend Your Money Where It Matters
  - Your Enemies Will

Solutions Based in Science

- Divide and Conquer
  - Not All Problems Are the Same
  - Fundamental Ability to Protect Information is Critical
  - TCB Isolates Security Controls
- Distinguish Between Policies
  - Some Enforced With High Assurance, Some Can't Be
  - Some Violations Result in Massive Loss – Some Do Not
  - Closed User Groups with MAC for Crucial Distinctions
- Don't Have to Solve All Hard Problems at Once
  - Spend Your Money Where It Matters
  - Your Enemies Will
  - Leverage MAC TCB On Selected Components

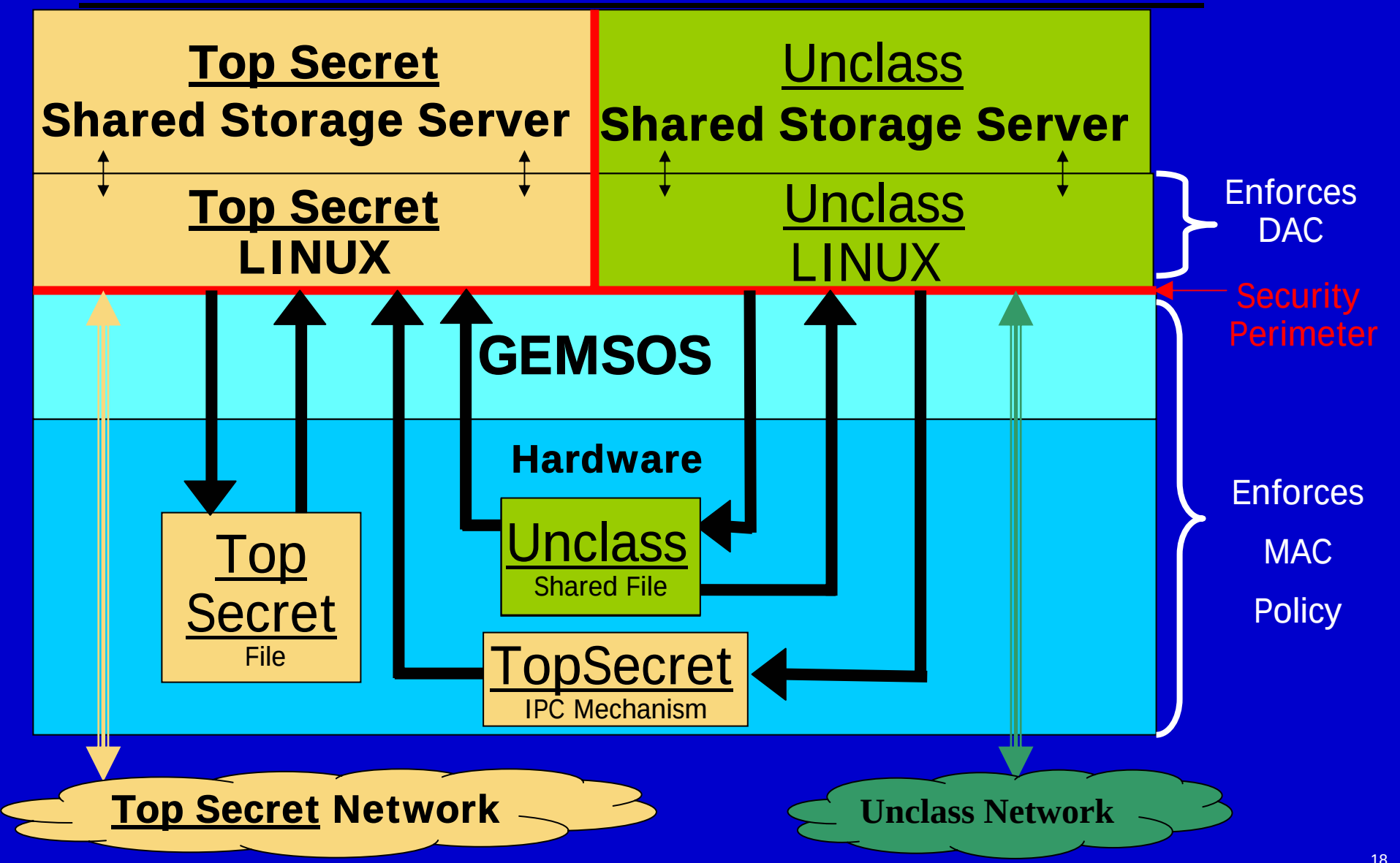
Solutions Based in Science
Layered Protection –"DAC on MAC"



Product Properties of Kernel that Ease Use

- Manageable multilevel data structures
  - Applications run at numerous security levels
  - Quantity of levels grows over time
- TCB subsets to extend TCB
  - Addition of trusted applications
  - Hierarchical protection domains (rings)
- Add trusted components without reevaluation
  - Add high assurance components, e.g., sanitization functions
  - Also components with balanced assurance, e.g., DAC policies
- No covert storage channels, e.g., IPC
- Dynamic unpredictable instantiation of subjects
  - Different security levels in same process
  - Critical for performance
- Deliberately designed as reusable “smart hardware”
  - Identical platform for range of applications

Example OEM Might Use Product: DAC on MAC storage architecture



Product Tools and Documentation

- System Generation Tools
  - Install new instance of System
  - Configuration and administration (security choices)
- Software Development Environment
  - Unix based cross development environment
  - Compilers and Linker
  - Testing and debugging environment
- End User Documentation
  - Trusted Facilities Manual
  - Security Features Users Guide

Key Secure Product Properties: Example Nominal Candidates

	High Assurance Policy Enforcement	Extensible Security	Inherent Layered Applications Support	Small Footprint	Demonstrated Evaluation Path	Field Proven
XTS-400		X	X			X
Boeing LAN	X			X	X	X
SE Linux		X	X			
Trusted Solaris		X	X			X
MILS (Partition Kernels)		X		X		
LaGrande		X		X		
GEMSOS	X	X	X	X	X	X

COTS Class A1 GEMSOS platform: Seasoned code, proven, versatile

Application	Customer	Details
Virtual Private Network (VPN)	National Security Agency	Blacker – performance; secure most sensitive national interests (Unisys)
Authentication Gateway	Pentagon	HSRP –10,000 users various clearances (Grumman)
Enterprise Automation	British Ministry of Defence	CHOTS – UK's largest military IT procurement (ICL)
Application Server & Clients	NATO	SACLANT – basis of security design (Contel)
Database	Oracle	Trusted Oracle 7 – designed to run on (Oracle)

- Commercial off-the-shelf (COTS) product being delivered today
  - Developer "Tool Kit": Pre-production platform, cross-development tools
  - Source code for worked MLS example and application libraries
- MLS "unhackable" web server demonstration running today
  - High assurance browse down – cross security domain transfer on server
  - Authoritative web pages cannot be defaced

Key Lessons

- Don’t be first with new security paradigms
  - Several successful worked examples of technology
  - Extensive peer review of science behind solutions
- Well specified stable evaluation criteria help
  - Largely knew what counts for success
  - Evaluator “interference” relatively minor
  - Formalization of policy vice formalization of mechanism
- Not rocket science
  - Good development standards strictly adhered to
  - Engineers adapt and thrive
  - Development team develops instead of debugs

Limiting Risks and Costs of Product Evaluation

- Verifiable Security Needs Third Party Assessment

- Third Party Evaluations can be Expensive and Risky
  - Build from scratch specifically to be evaluated can take years
  - Need established criteria, new protection profile
  - Once finally built, system must be evaluated against criteria

- Mitigate Costs and Risks Through Use of Proven Methods
  - Build on existing product that has been evaluated (e.g., Class A1)
  - Take advantage of existing criteria with proven worked examples
  - Effective use of previous evaluation – ratings maintenance

æSec Corporation

The power of verifiable protection™

Creating High Assurance for a product: Lessons Learned from GEMSOS

Roger Schell
Roger.Schell@aesec.com
(831) 657-0899

IEEE IWIA 2005
College Park, MD
March 23, 2005