z/OS System Integrity:

How z/OS Software Enables You to Protect Your Assets

Scott Fagen – Distinguished Engineer

Chief Architect: z Systems and Workload Automation

CA Technologies

scott.fagen@ca.com

#SHAREorg

SHARE is an independent volunteer-run information technology association that provides education, professional networking and industry influence.

Trademarks

Full disclosure:

- The presenter is an employee of CA
- Products and subsystems mentioned in this presentation are illustrative and do not pretend to be comprehensive. While I would like for you to buy CA products, I acknowledge that there are many ways to skin the integrity cat. In most cases I've tried to include products with similar purposes from CA and IBM.

The following are trademarks of CA:

- CA, CA ACF2, CA Allocate, CA Disk, CA MIM, CA NetMaster, CA Top Secret

The following are trademarks of EMC:

- EMC, SRDF

The following are trademarks of IBM:

- CICS, DB2, DFSMS, DFSMShsm, FlashCopy, GDPS, HiperSockets, IBM, IMS, MVS, OS/390, Parallel Sysplex, PR/SM, RACF, System z, Websphere, z Systems, z/OS

Integrity

- The goal of this presentation is to introduce you to the principles of system integrity for z Systems and z/OS and the division of responsibilities for delivery of integrity on the platform

- I hope that this discussion helps you to define the levels of integrity required for your installation and how you might go about delivering that integrity

- The American Heritage Dictionary says;
  - Steadfast adherence to moral or ethical code
  - The state of being unimpaired; soundness
  - The quality or condition of being whole or undivided; completeness

- This is a good intuitive definition, how does that relate to a computer system?

z/OS Statement of Integrity

From http://www-03.ibm.com/servers/eserver/zseries/zos/racf/zos_integrity_statement.html

First issued in 1973, IBM's MVS™ System Integrity Statement, and subsequent statements for OS/390® and z/OS, has stood for over three decades as a symbol of IBM's confidence in and commitment to the z/OS operating system. IBM reaffirms its commitment to z/OS System Integrity.

IBM's commitment includes design and development practices intended to prevent unauthorized application programs, subsystems, and users from bypassing z/OS security – that is, *to prevent them from gaining access, circumventing, disabling, altering, or obtaining control of key z/OS system processes and resources unless allowed by the installation*.

Specifically, z/OS "System Integrity" is defined as the inability of any program not authorized by a mechanism under the installation's control to circumvent or disable store or fetch protection, access a resource protected by the z/OS Security Server (RACF®), or obtain control in an authorized state; that is, in supervisor state, with a protection key less than eight (8), or Authorized Program Facility (APF) authorized. In the event that an IBM System Integrity problem is reported, IBM will always take action to resolve it.

IBM's long-term commitment to System Integrity is unique in the industry, and forms the basis of z/OS' industry leadership in system security. z/OS is designed to help you protect your system, data, transactions, and applications from accidental or malicious modification. This is one of the many reasons IBM System z™ remains the industry's premier data server for mission-critical workloads.

CA Integrity Statement

http://www.ca.com/us/collateral/technical-documents/na/ca-technologies-mainframe-products-integrity-statement.aspx

CA Technologies offers this integrity statement to document our confidence in our products. This confidence stems from a comprehensive development process with extensive research, planning, coding, and testing.

Beginning with the first release of each product, we have developed all releases in compliance with IBM requirements in the IBM z/OS® System Integrity Statement, which can be found on the IBM website.

We are firmly committed to the quality and integrity of its mainframe products, by providing products and services that meet or exceed our external and internal customer requirements and to continually strive to improve our efficiency and effectiveness in doing so. We have developed, implemented, and continue to improve programs to ensure the quality and integrity of our products.

Types of Integrity

- **Hardware integrity:** always getting a predictable result based on the rules in the hardware architecture
- **Software integrity:** enforcing the installation's policies by controlling access to and usage of hardware facilities, so users can do only what is allowed them
- "z/OS System Integrity" is a subset of software integrity and it depends mainly on key controlled protection
  - definitions coming up
- Security, data integrity and functional integrity are more encompassing ideas and we will touch on them in the 2nd half of this talk

Understanding Architecture

- Hardware architecture – “What the processor does”
- Software architecture – “How software uses it”
  - Layered on top of the hardware architecture
  - Some software constructs must line up with hardware
    - Address space implementation, linkage stack, cross-memory etc. largely invisible to ordinary users
  - Other software constructs are simply software
    - Jobs, tasks, programming interfaces etc.
    - These are the things you are used to seeing
- System, subsystem configuration and deployment

Integrity Delivery Continuum

Hardware Architecture	Software Architecture	Carbon Based Units (System Programmers, Security Administration)
• Described (but not limited) by POO • Defines the constructs for enabling system integrity: • Problem/Supervisor State • Storage Keys • Addressing modes • Instructions	• Described[1] by provided documentation • OS, Middleware • Builds on hardware architecture to provide the context and services to manage system integrity • Provides enablement to support application and data continuity	• Define policies: • Security • Authorization • Application availability • Data resilience • Implement installation architecture to support policies
Examples: SVC, PC, SPKA, LPSW, ECKD architecture, Data Replication	**Examples:** APF, PPT, STOKEN, TTOKEN, SAF/Security Product, Sysplex, Data Backup, Data Replication Services, TLS, IDS	

[1] This documentation is arguably incomplete and subject to change at the whim of the software vendor

Hardware Basis for Integrity

- Physical memory and processor integrity
  - In the 1960s, hardware used to be flaky, so IBM engineers set out to make the System/360 as reliable as possible through self checking and recovery
    - Error Detecting Code (EDC) and Error Correcting Code (ECC) circuitry detect certain memory and CPU errors and correct a subset of those without disruption to software
  - Fail fast philosophy; don't spread the misery:
    - Machine check interrupt issued for non-recoverable errors
    - Machine-check recovering processing drives memory sparing and alternate CPU recovery

Hardware Basis for Integrity

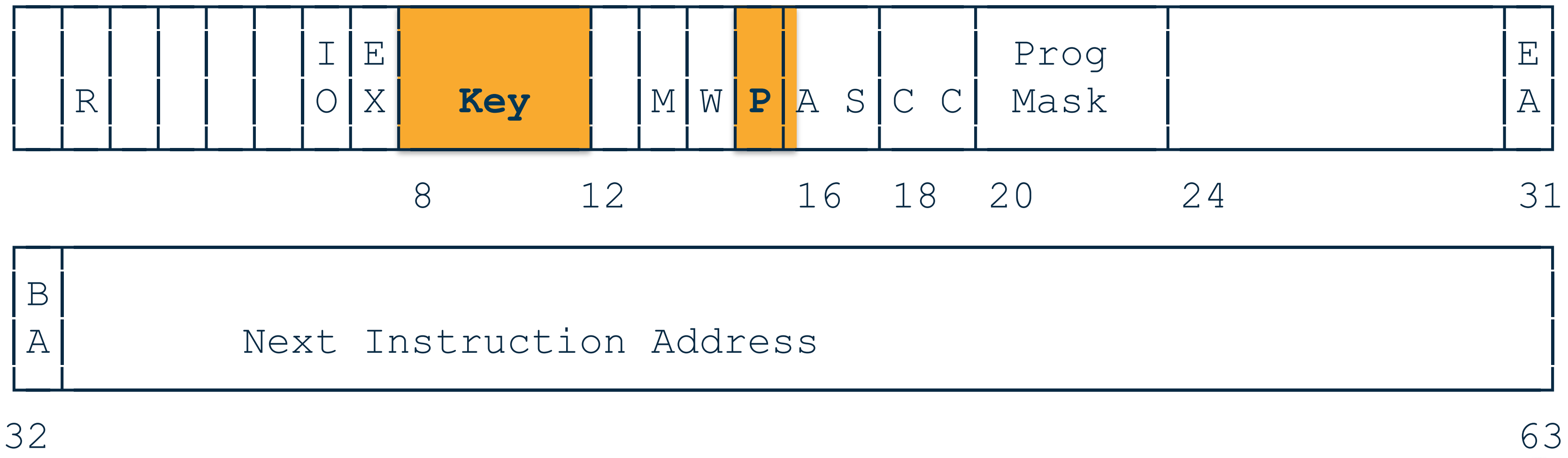
- Key controlled memory access
  - Detailed discussion coming up next
- There are many more hardware-based integrity features, some of which will be discussed after we discuss the foundations for z/OS System Integrity
  - CPU sparing and alternate CPU recovery
  - LPAR management
  - I/O subsystem and channel architecture
  - More

Supervisor State vs. Problem State

- Critical hardware design point: segregate system and user functions based on their state:
  - Supervisor state is for operating system and other systems software (e.g. security subsystems, performance monitors)
  - Problem state is for ALL user programs
- The hardware prevents problem state programs from executing supervisor instructions – ABENDS0C2
- The operating system and other system software must also play their part to prevent users from gaining the ability to violate these controls

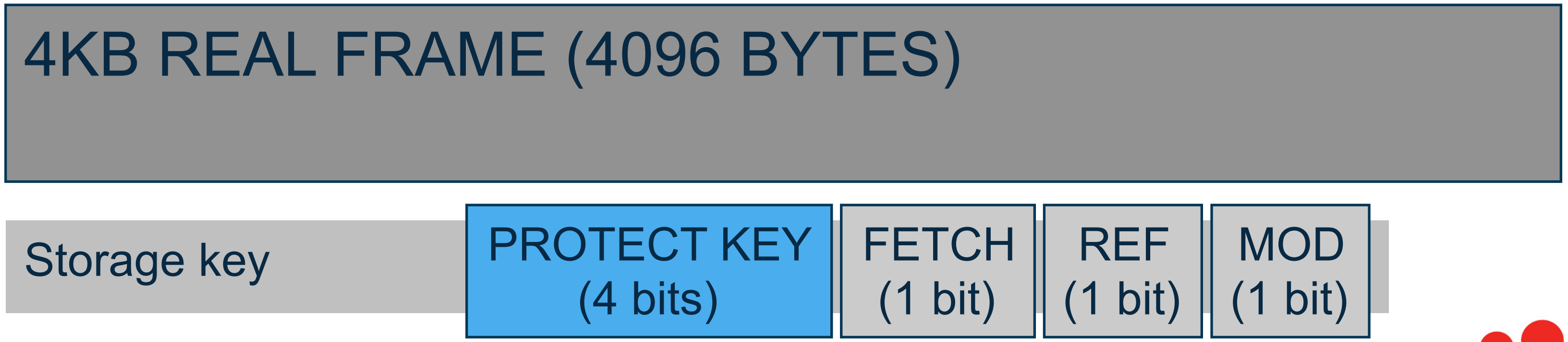
Program Status Word (PSW)

- PSW is a special state "register" in each CPU:
  - PSW and registers are saved as "state" for any running work
  - OS loads PSW and registers to dispatch work
  - PSW key controls access to memory (coming up next)
  - Problem state bit ON disallows all privileged instructions



Hardware (Memory) Storage Key

- This is a 7 bit data area associated with each 4KB frame:
  – The storage key is logically associated with the frame, but it is not part of addressable (real) memory
  – All bits are initialized by operating system software
    - Set as a unit by special instructions (ISKE/SSKE)
    - Ref and mod bits set by hardware on memory access
  – IVSK returns current value of protect key and fetch bit



Key Controlled Protection

For every memory reference (fetch or store) the CPU compares the PSW key with the memory protect key:

- If PSW key is zero, fetch and store access is allowed
- If both keys match, fetch and store access is allowed

- If keys don’t match, but fetch bit is OFF, fetch access is allowed, but store denied with ABENDS0C4-4

- If keys don’t match, and fetch bit is ON, all access is denied with ABENDS0C4-4
- Storage protection feature (e.g. for low PSA) trumps all keys. If set, any store attempt gets ABENDS0C4-4

Changing PSW Keys

- The SPKA instruction changes the current PSW key:
  - This is a semi-privileged instruction
- Supervisor state programs can switch to any key:
  - They should use system keys (0-7) for most of their work, problem keys (8-9) need special care
- Problem state programs can only switch to a key that is allowed by their Protect Key Mask (PKM):
  - PKM is set up in control register 3 by the OS
  - z/OS gives unauthorized programs access to key 8 and key 9 ONLY

Virtual Memory Is The Firewall

- Virtual memory segregates users from each other
  - Address Spaces (and subspaces although they are rarely used) are containers for user programs and data:
    - Users can’t see or touch anything outside of their own address space, without the overt co-operation of an authorized program
    - But, common storage can be problematic
    - Fallout from most program failures in a user address space is limited to that address space, no harm to others
  - Data Spaces and shared memory objects offer hardware/ operating system mediated, controlled sharing of data

Definition of “Authorization” in z/OS

- An authorized program is one that;
  - Is executing in supervisor state and/or
  - Is executing in a system PSW key (0-7) and/or
  - Has a protection key mask that would allow it to execute in a system PSW key (0-7) and/or
  - Is running under an APF-authorized job step task (definition of APF coming up shortly)

Hardware/Software Boundaries

- z/OS controls access to the hardware mechanisms for altering the authority level of the running program:
  - PKM controls which keys a problem program can use
  - SVC or PC instructions switch to supervisor state and branch to addresses defined in SVC and PC tables:
    - PC entry table definitions define who can call authorized functions
    - SVC logic can also limit access to authorized functions
- Problem (user) programs can only call authorized programs through state change interfaces (PC/SVC)

Software Provided Assists

- How does z/OS restrict access to these functions?
  - It only allows “authorized” programs to request the services that create or alter protection key and state in the first place
  - Gives rise to the chicken and egg problem; how does a program become authorized in the first place?
- Through control statements in SYS1.PARMLIB members:
  - Program Properties Table (SCHEDxx)
  - APF List (PROGxx)
- Authorized programs must be fetched from authorized libraries:
  - Nucleus, LPA, linklist or any APF library

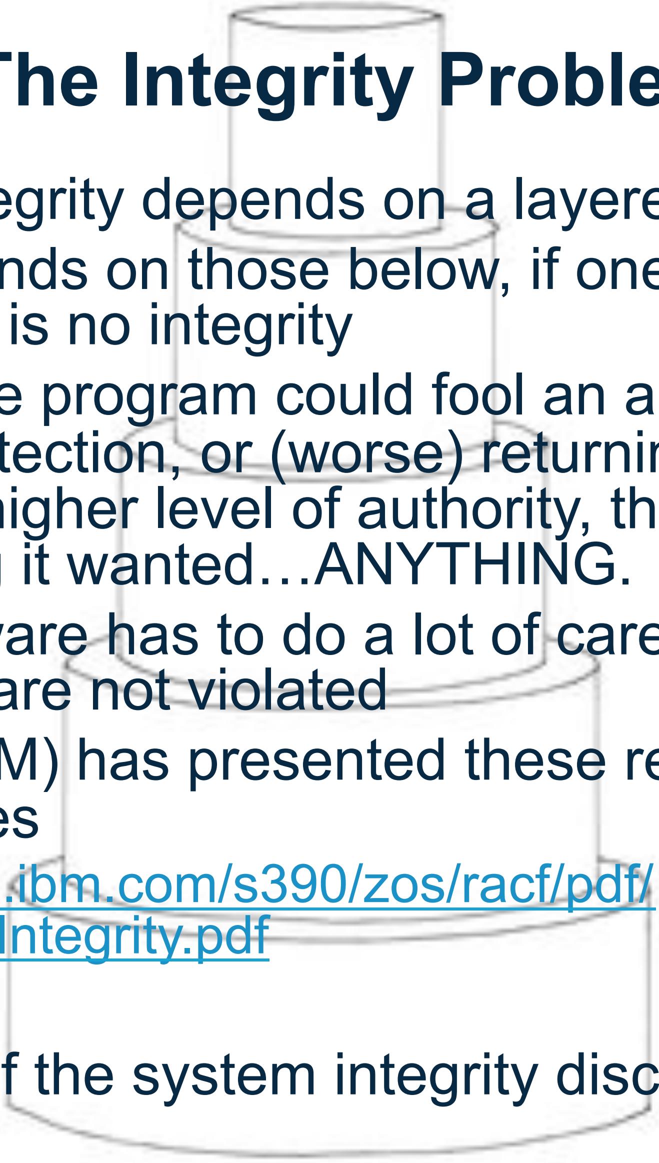
APF Authorization

- The Authorized Program Facility (APF):
  - The installation (system programmer) designates all libraries considered to be authorized
  - APF list is defined in SYS1.PARMLIB(PROGxx):
    - IEAAPFxx member is a deprecated interface
  - Dynamically modifiable via the SETPROG command
  - Certain system libraries are automatically presumed to be authorized (nucleus, linkpack, linklist)
- Authorized programs can only be loaded from an APF authorized library – ABENDS306 otherwise

What Is An APF Authorized Job?

- Job step programs are considered “authorized” if they are loaded from an APF library and linked with AC(1):
  - Checked once at job step initiation time only
  - JSCBAUTH is set to 1 if both are true, else set to 0
  - Every new subtask is also considered authorized
- Authorized job step gets control in problem state and PSW key 8, but is allowed to issue the MODESET SVC:
  - ABENDS047 issued if job step is not authorized
- AC(1) is meaningless except for job step programs:
  - Should NEVER be specified otherwise

The Heart Of The Integrity Problem!



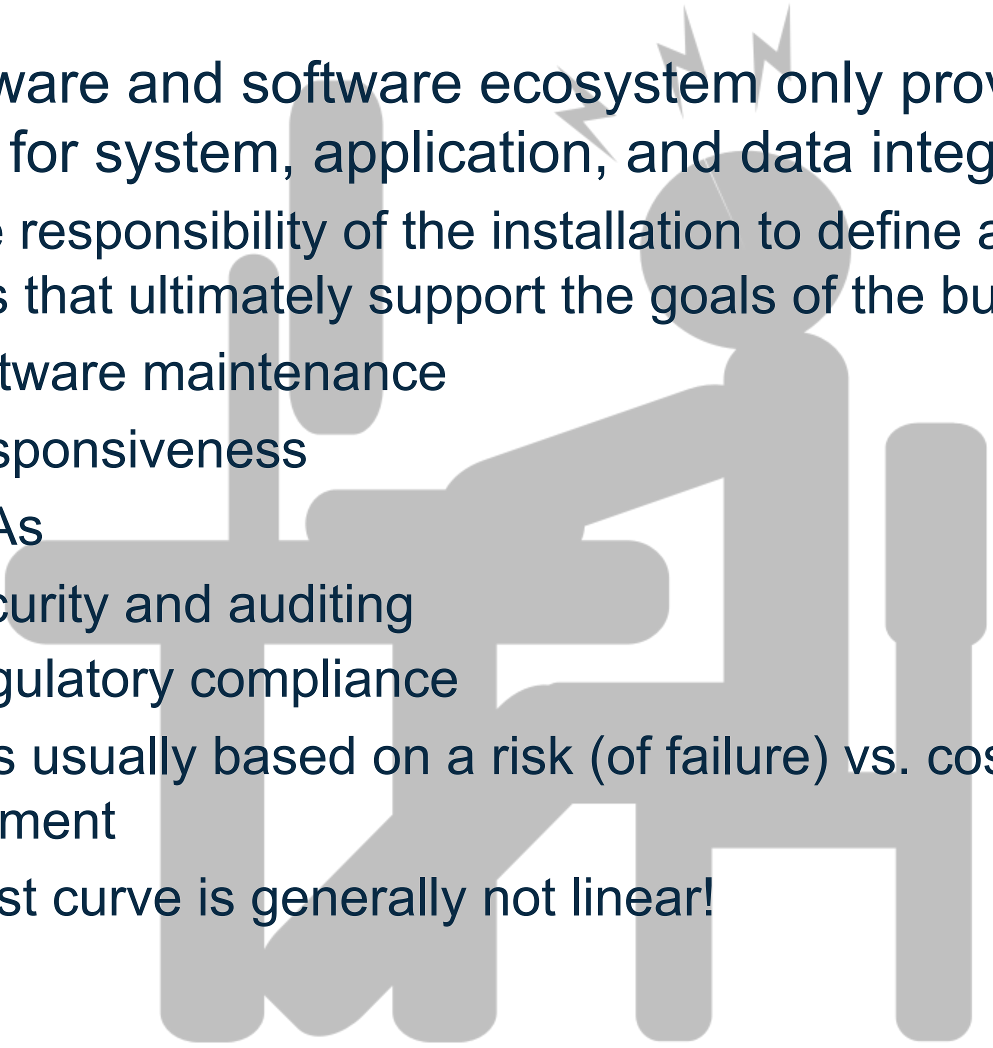
- z/OS system integrity depends on a layered model
- Each layer depends on those below, if one of those layers is faulty then there is no integrity
- If a problem state program could fool an authorized program into violating key protection, or (worse) returning the problem state program with a higher level of authority, then that program could then do anything it wanted…ANYTHING.
- Authorized software has to do a lot of careful checking to ensure that these rules are not violated
- Karl Schmitz (IBM) has presented these requirements in a number of venues
  - ftp://public.dhe.ibm.com/s390/zos/racf/pdf/zOS_System_Integrity.pdf

- This is the end of the system integrity discussion

The Rest Of The Story

- System integrity as just defined is necessary for correct operation of a z/OS system, but that alone is not enough:
  - Security controls are needed
  - Data integrity mechanisms are needed
  - Overall functional integrity and availability mechanisms are needed
- The installation is responsible for defining and managing the policies that z/OS uses to protect your corporate assets:
  - Failure to be vigilant in these policies is a recipe for disaster, could result in anything from an outage to stolen, lost, or corrupt data

Carbon Based Integrity



- The hardware and software ecosystem only provide the *capability* for system, application, and data integrity:
  - It is the responsibility of the installation to define and implement the policies that ultimately support the goals of the business:
    - Software maintenance
    - Responsiveness
    - SLAs
    - Security and auditing
    - Regulatory compliance
  - Policies usually based on a risk (of failure) vs. cost (to implement) assessment
  - The cost curve is generally not linear!

Minimal Requirements

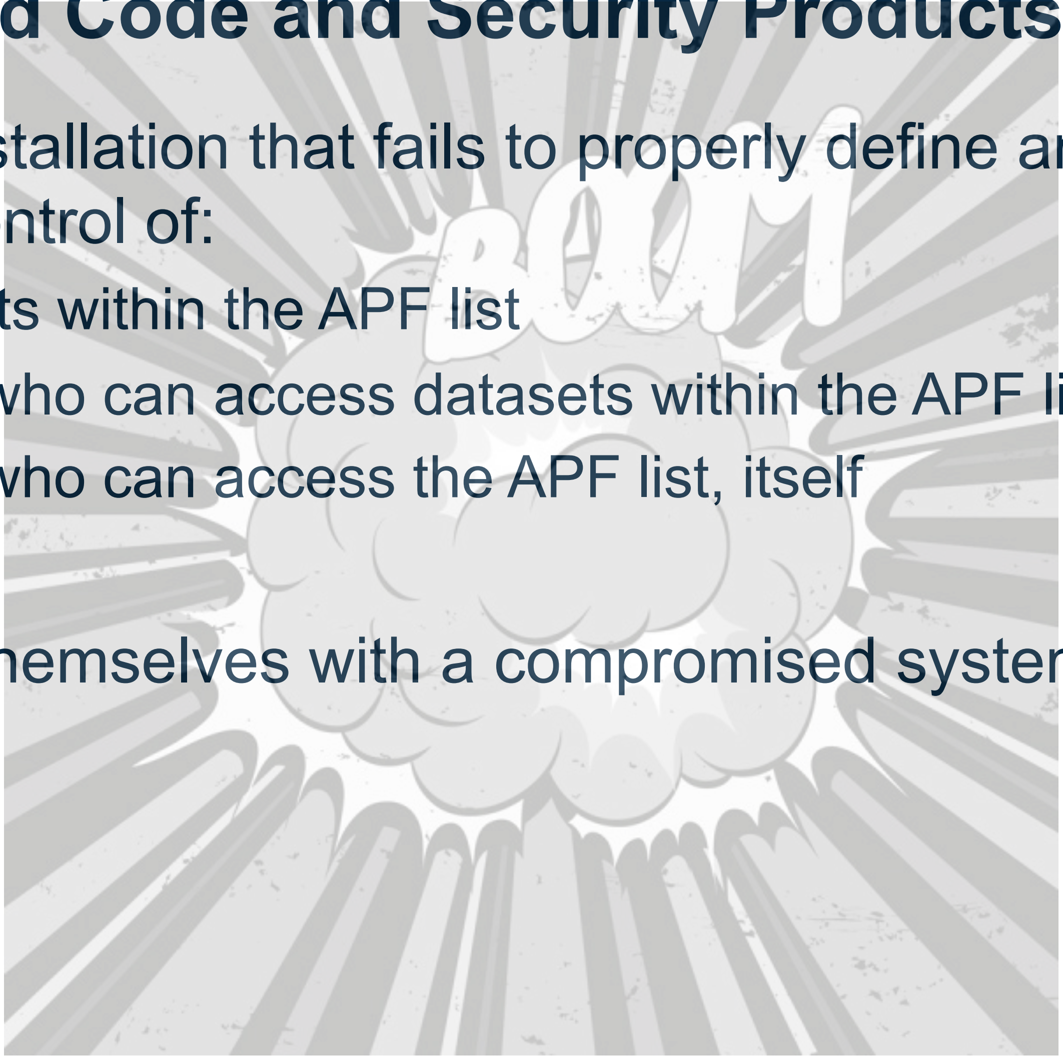
- By default, the operating system, middleware, and hardware provide a great deal of protection, but the installation is required to specify the policies for authorization, identity, and access to resources:
  - Authorized libraries
  - Users/Groups
  - Secured resources
  - Network implementation and security policy

- *The settings that come 'out of the box' are insufficient to maintain even the appearance of integrity*

Authorized Code and Security Products

- Beyond APF, z/OS provides SAF (Security Authorization Facility) interfaces to answer the question:
  "Can User X, running Program Y, access Resource Z?"
  - Code runs as an authorized extension of the OS and is called by the OS at various system control points, such as OPEN
- Your installation chooses which security product(s) realize these interfaces (RACF, CA Top Secret, CA ACF2):
  - Essentially they are just the query and management interfaces to a database of product and installation defined names and rules:
    - Resource names, Class names, User and Group names
    - Levels of authority
    - A cross-reference mapping users/groups authority to resources

Authorized Code and Security Products

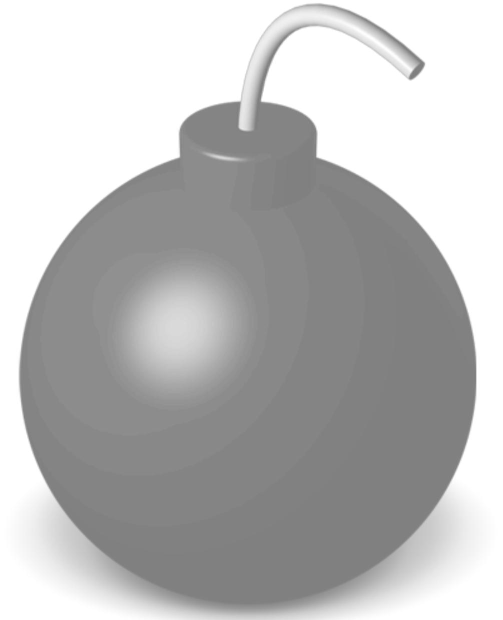


- So, an installation that fails to properly define and maintain secure control of:
  - Datasets within the APF list
  - Users who can access datasets within the APF list
  - Users who can access the APF list, itself

- can find themselves with a compromised system!

Authorized Code and Security Products

- On a system that lacks proper Carbon Based integrity:
  - An unsophisticated user can unintentionally deny service to, or damage or delete key resources
  - A unsophisticated, malicious user can likely do the same as the unsophisticated user, except that he may do so *intentionally*



Authorized Code and Security Products

- An educated, malicious user can wreak havoc on the system:
  - A user who can get an *arbitrary* program into supervisor state/key 0 essentially owns the “keys to the kingdom”
  - Depending on their level of expertise, the malicious user can delete or corrupt data <u>and</u> cover his tracks! He could even impact running programs through alteration of those programs and/or data in memory.

Data Storage Integrity

- I/O Integrity features:
  - ECKD architecture and channel subsystem:
    - Serialize access to external storage at physical level
    - Set File Mask CCW prevents accidental reads or writes outside of extent boundaries on disk – file integrity
    - Channel programs prevent I/O outside area designated for each I/O – avoids I/O overlays
  - RAID architectures, virtual tape systems:
    - Make multiple copies of data on multiple disks
    - Detect and prevent/enable recovery from certain classes of errors

Data Content Classification

- Current z/OS data security is limited to mapping the names of the resources to a set of rules controlling access
- New regulations are requiring security controls based on the data within data sets (e.g. personally identifiable information)
  - Who can see what (and when)
  - Encryption of data at rest (in storage)
  - Encryption of data in motion (transmission)
  - Acceptable location for data at rest
  - Acceptable location for data in motion
- These regulations are continuously in flux and vary by country
- Locating and classifying data can be a time consuming task

Network Integrity

- I’m not really a networking guy, but…
- Data flowing over a real connection network may need to be secured (encrypted):
  - IPSec, SSL, TLS, SSH
- Is the system vulnerable to an attack?
  - Firewalls?
  - Check TCP/IP stacks, ports
  - Are you using IDS (Intrusion Detection Services)?
  - Are you monitoring? CA NetMaster, NetView
- There are always good IBM presentations at SHARE discussing securing network around z Systems

Network Integrity

- Take advantage of z Systems and z/OS “exclusives”:
  - Extinction solution: nothing to snoop or sniff
  - HiperSockets:
    - Enables communications between images in the same hardware footprint, even z/OS to z/VM or zLinux
    - No need to encrypt or secure data flowing over these connections:
      - Memory to memory data move (through PR/SM)
      - Cannot be spoofed or intercepted
  - TCP/IP over XCF:
    - Enables TCP/IP connections between images in the same sysplex
  - RoC/E
    - Enables communications between multiple z Systems images via a high-speed, private interconnect

Keeping the environment whole

- *Integrity:*
  - *The state of being unimpaired; soundness*
  - *The quality or condition of being whole or undivided; completeness*

- What does it take to achieve higher levels of integrity (a/k/a "availability") than provided by a single z/OS image?
  - The costs of adding the next "9" of availability are not linear!

Improving System Availability
"Resource Sharing"

- Originally, Sysplex was envisioned as a means to achieve capacity requirements during the transition from bipolar to CMOS processor technology – Now it is a means to help ensure continuous availability and maximize resource utilization:
  - Simple numbers game:
    - In a single system environment – outage of one machine =
      100% capacity outage
    - In a 32-way environment – outage of one machine = 3% capacity outage
  - This capacity can be redeployed to other LPARs while the system is being recovered

Improving System Availability "Resource Sharing"

- Sysplex components/middleware transparently help you manage capacity and shared resources:
  - **zOS:** XCF/XES, WLM/IRD, GRS, IOS, DFSMS, Allocation, Console, Logger, TCP/IP
  - **Network:** VIPA, DVIPA ([Dynamic] Virtual IP Address)
  - **Subsystems:** CA MIM, MQSeries, CA Allocate, CA Disk, CA NetMaster

Improving System Availability "Resource Sharing"

- Even in a single CPC environment, Resource Sharing can help achieve higher levels of availability:
  - Operating System (and major subsystem) instances DO occasionally fail (more frequently than hardware)
  - LPAR is no longer optional, why not take advantage of it?
  - Parallel Sysplex is now well established technology at most large z/OS installations

Improving Application Availability Data Sharing

- Once you've decided to move to a sysplex environment, you can improve application availability by enabling them to run on any system within the sysplex (cloning):
  - **Application Server:** CICS, IMS, WebSphere Application Server
  - **Database Server:** CA Datacom, CA IDMS, DB2
  - **Messaging:** TCP/IP, Websphere-MQSeries
- Two different schools of thought
  1. Fast failover/shadowing (CA Datacom)
  2. Data sharing (CA Datacom, CA IDMS, DB2)

Improving Data Integrity Backup/Archive

- Simplest and most popular method for maintaining a level of data integrity
- Periodically copy, compress, and offload data
- CA-Disk, DFHSM, FlashCopy
- Can archive to another location for further protection
- Protects against media failure
- Provides recovery up to last backup
  - Some data loss is inevitable
  - Requires that backup copies are made at the right time(s) to ensure recoverability (consistency point)

Improving Data Integrity Backup/Archive

- When data are backed up or archived on tape, the data can become “separated from your control”
  - Does your data contain:
    - Trade secrets
    - Personal data
    - ?
- PTAM has a nasty habit of leaving tapes on the side of the road
  - Maintaining the integrity of this data may require encryption:
    - CA Tape Encryption, Encryption Facility for z/OS, Encrypting Tape Drives
    - However, encryption adds a new set of complexities

Improving Data Integrity
Continuous Data Protection

- A scheme made viable by reduced cost of storage media and fabric
  - When a write occurs, change the ‘bits on disk’
  - Also, maintains a log of all changes:
    - [Looks a lot like database recovery]
  - Can now rollback any and all writes (transactions) by going through the update log
  - Can cut periodic backups to clip the log at particular intervals
- Implementations can be on a storage controller, volume or dataset/file level
- Can eliminate the need for point-in-time backups
- May also still require a mirroring solution (next)

Improving Data Integrity
Storage Mirroring

- Define secondary copies of all important data for replication
- All writes are “duplexed” to primary and secondary controllers:
  - Synchronous – write not complete until both units confirm complete:
    - No data loss when switching to secondary
  - Asynchronous – write complete when primary confirms complete; secondary updated periodically:
    - Small amount of data loss when switching to secondary

Improving Data Integrity
Storage Mirroring

- Periodic backup/archive or CDP required to defend against application errors and/or malicious damage to data:
  - Storage controllers can't distinguish between a 'good' write and a 'bad' write
- Defends against controller failure and can be the basis for disaster recovery/business continuity solutions
- Requires ongoing diligence to ensure recoverability (appropriate consistency groups)

Maximum Insurance
Multi-Site Parallel Sysplex

- For applications and data that require maximum availability (nearly zero downtime), an installation can implement a Multi-site Parallel Sysplex:
  - GDPS (Geographically Dispersed Parallel Sysplex)
  - GDDR (Geographically Dispersed Disaster Restart)
- Replicates all hardware (z Systems, storage, network) and software at both sites with additional fiber links between the sites:
  - Recommend two independent routes for maximum availability
- Distances < 50km – can implement synchronous mirroring; for longer distances, asynchronous mirroring is recommended:
  - Speed of light is a limiting factor
- Highly expensive, but necessary when regulatory requirements dictate very short MTTR with nearly zero data loss:
  - <1 hour can be achieved under right conditions
  - <1 day in most cases

Maximum Insurance
Multi-Site Parallel Sysplex

- Complete failure of primary site can be mitigated by switching over to secondary site
  - If distances are great enough, business computing can continue through a nuclear strike (!)
  - Your users may be a little distracted, though
- Requires a great deal of operational rigor:
  - Utilizes latest and greatest technology in z Systems
    - Parallel Sysplex/CF Duplexing
    - Capacity On Demand
    - Hiperswap
    - Replication (PPRC, XRC, SRDF)

Maximum Insurance
Multi-site Active-Active Applications

- RPO and RTO can be further improved for applications that can be enhanced to deal with cross-site software based database replication
- Application data are replicated in multiple sites
  - Rather than replicating the bitwise changes to storage, completed transactions are packaged and transmitted to the other site to update the local copy of the database
  - A process called "Change Data Capture" (CDC)
  - Improves the coherence of data
  - With appropriate Server/Application State Protocol (SASP) networking gear, an installation can run the same application at two sites separated by long distances
- Three levels:
  - Active-Standby (nearly no data loss, RTO near zero)
  - Active-Query (nearly no data loss, RTO near zero, alternate sites can process query transactions)
  - Active-Active (nearly no data data loss, RTO near zero, all workload can be distributed over multiple sites

Summary

- System integrity is not something that z Systems and z/OS deliver “right out of the box”
- Maintaining integrity requires careful consideration and periodic re-evaluation as the system, software stack, and business requirements change – it may even be true that different parts of the business have needs for differing levels integrity
- z Systems, z/OS, middleware, and your installation work together to provide integrity
- There are a wide range of integrity options available when implementing z Systems and z/OS

Questions?

Complete your session evaluations online at www.SHARE.org/Seattle-Eval

Complete your session evaluations online at www.SHARE.org/Seattle-Eval