

Securing Your z/TPF Environment

Mark Gambino

Jamie Farmer

Angel Baez

Jennifer Chiarieri

Claire Durant

Chris Filacheck

Luanne Jagich

Kelly Paesano



Cloud

IBM Z

IBM
®

Redpaper



IBM Redbooks

Securing Your z/TPF Environment

July 2024

Note: Before using this information and the product it supports, read the information in “Notices” on page vii.

First Edition (July 2024)

This document was created or updated on July 29, 2024.

© Copyright International Business Machines Corporation 2024. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	vii
Trademarks	viii
Preface	ix
Authors	ix
Now you can become a published author, too!	x
Comments welcome	x
Stay connected to IBM Redbooks	xi
Chapter 1. Introduction.....	1
1.1 Getting started.....	3
Chapter 2. Security by design with risk management frameworks	5
Chapter 3. Understanding and reducing risk with the NIST Cybersecurity Framework 9	
Chapter 4. Identifying your cybersecurity risks.....	11
4.1 Threat target analysis	12
4.2 Regulation adherence	12
4.3 Compliance reporting on z/TPF	13
Chapter 5. Protecting your z/TPF data and resources.....	17
5.1 Foundational protection	18
5.1.1 Maintaining currency	18
5.1.2 Securely installing software and firmware	18
5.1.3 Future proofing your protection.....	21
5.1.4 The z/TPF keystore.....	22
5.2 Data protection	31
5.2.1 Securing data in flight	31
5.2.2 Securing data at rest	37
5.2.3 Protecting data in use	45
5.2.4 Protecting your data from corruption.....	48
5.3 Access control.....	51
5.3.1 Leveraging firewalls	51
5.3.2 Designing and managing access to services	52
5.3.3 Controlling access to data for z/TPF Java applications	53
5.3.4 Managing access for z/TPF support for MongoDB	53
5.3.5 Controlling access to data with z/TPF support for MongoDB	55
5.3.6 Controlling access to data with z/TPF data events	56
5.3.7 Managing access to the z/TPF file system	57
5.3.8 Managing access for z/TPF operators	58
Chapter 6. Detecting anomalies and security events	59
6.1 Data corruption detection	60
6.2 Workload overload detection and prevention	64
6.2.1 Preventing excessive network connections	64
6.2.2 Preventing excessive application traffic	64
6.2.3 Preventing excessive service requests.....	65
6.2.4 z/TPF REST throttling	66
6.2.5 Preventing system overload	68

6.2.6 Monitoring network anomalies	69
6.3 Resource overload detection and prevention	71
6.3.1 Monitoring z/TPF ECB resources	71
6.3.2 Monitoring resources using Runtime Metrics Collection	72
6.4 AI-infused detection	72
Chapter 7. Responding and recovering from security events	75
7.1 Security incident response using cryptographic agility	76
7.2 Security incident response with access controls	76
7.3 Data corruption recovery	77
7.4 Problematic application and system code recovery	81
7.5 System outage recovery	84
7.6 Site failure recovery	84
Chapter 8. Conclusion	87
Appendix A. z/TPF system and workload resiliency	89
A.1 Transaction isolation	90
A.2 Memory protection	91
A.3 Memory validation	92
A.4 Program service restriction	92
A.5 Database isolation	92
A.6 High availability in a Java environment	93
A.7 Workload spike protection	94
A.8 HA and hardware redundancy	95
Appendix B. Identify your threat targets	97

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, MD-NC119, Armonk, NY 10504-1785, US

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at <https://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks or registered trademarks of International Business Machines Corporation, and might also be trademarks or registered trademarks in other countries.

DS8000®

FlashCopy®

IBM®

IBM API Connect®

IBM Z®

IBM z16™

Redbooks®

Redbooks (logo)  ®

System z®

z/OS®

z15®

z16™

The following terms are trademarks of other companies:

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other company, product, or service names may be trademarks or service marks of others.

Preface

In September 2023, IBM published *Accelerate Mainframe z/TPF Application Modernization with Hybrid Cloud*, 5714-00 that details IBM z/Transaction Processing Facility (z/TPF) modernization techniques in a high-performance hybrid cloud environment. Because of the financial impacts and brand reputation damage that can result from cyberattacks on your business, modernization of your z/TPF environment must be done with security at the forefront. This Redbook provides information about securing and enhancing the cybersecurity features of your z/TPF systems, ensuring both current and future needs are met.

Authors

This paper was produced by a team of specialists from around the world working at IBM Redbooks, Poughkeepsie Center.

Mark Gambino is an IBM® Distinguished Engineer and Chief Architect of the z/TPF portfolio. He has expertise working with real-time transaction processing environments in the Travel and Transportation, Hospitality, and Financial Services sectors that require high performance and the ability to scale to hundreds of thousands of secure transactions per second, while maintaining true high availability. His extensive technical depth includes networking, middleware, database, task dispatching, memory management, security, and other operating system services, plus problem determination, and monitoring. Mark has enabled his products and customers to evolve, grow, and tackle the challenges of application modernization, creation of efficient hybrid cloud architectures, and real-time analytics.

Jamie Farmer is a senior software engineer. He has 27 years of experience in z/TPF. He holds a masters degree in computer science from Marist College. His areas of expertise include z/TPF security, networking, and middleware.

Angel Baez is a software developer. He has 4 years of experience in z/TPF. He holds a masters degree in cybersecurity from NYU Tandon. His areas of expertise include z/TPF security, networking, and middleware.

Jennifer Chiarieri is an advisory software engineer. She has 33 years of experience in z/TPF and z/OS® software development, software test, and information development. She holds a bachelors degree in Computer Science from Mount Saint Mary College. Her current areas of expertise include the z/TPF mail service, z/TPF secure file transfer, and the z/TPF automated test framework.

Claire Durant is a software engineer. She has five years of experience working with z/TPF. Her areas of expertise include z/TPF database access middleware, POSIX-compatible file system implementation, and message streaming infrastructure.

Chris Filacheck is a Senior Technical Staff Member (STSM) and the Database & Storage Architect for the z/TPF portfolio. He holds a masters degree in computer science from the University of Pittsburgh and has over 25 years of experience designing and developing z/TPF solutions that meet and exceed the performance, scalability, and high availability needs of z/TPF clients. His expertise includes many areas across z/TPF, including z/TPFD (the z/TPF high performance database solution), file systems, business events, DASD support, copy services, runtime environments, locking and serialization, and other operating system services.

Luanne Jagich is the Program Director of the z/TPF lab. Upon graduating from Penn State, she has dedicated over 30 years of experience to many aspects of System z®, including cryptographic hardware, zOS software, and channel/IO firmware. Additionally, she is a member of the TPFUG board of directors and has extensive experience in client care. She has previously authored papers on cryptography and application modernization.

Kelly Paesano is a product manager in the US. She has 9 years of experience across hardware and software at IBM. She holds a masters degree in engineering from North Carolina State University. Her areas of expertise include product design and delivery.

Thanks to the following people for their contributions to this project:

Jamie Roszel, Editor, IBM Redbooks®
IBM RTP

Lydia Parziale, Certified IT Specialist, PMI Certified Project Manager, IBM Redbooks Project Leader
IBM Redbooks, Poughkeepsie Center

Amanda Staples, Manager z/TPF
z/TPF, Poughkeepsie Center

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an IBM Redbooks residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:
ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our papers to be as helpful as possible. Send us your comments about this paper or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

ibm.com/redbooks

- ▶ Send your comments in an email to:
redbooks@us.ibm.com
- ▶ Mail your comments to:
IBM Corporation, IBM Redbooks
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- ▶ Find us on LinkedIn:
<https://www.linkedin.com/groups/2130806>
- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:
<https://www.redbooks.ibm.com/subscribe>
- ▶ Stay current on recent Redbooks publications with RSS Feeds:
<https://www.redbooks.ibm.com/rss.html>



Introduction

This chapter discusses the importance of cybersecurity and the value of choosing z/TPF and IBM Z to provide the data and system protection businesses need. With increasing volumes and complexity of cyberattacks driven by technologies like artificial intelligence and quantum computing, organizations face significant risks. To reduce these risks, it is crucial to implement proactive protection measures beyond basic requirements. IBM Z systems, known for their robust security, offer advanced capabilities to safeguard against both external and insider threats. These include data protection methods, cryptographic agility, system protection, and resiliency enhancements. By adopting IBM z/TPF and IBM Z, businesses can achieve comprehensive security and resilience, backed by established standards and frameworks.

Digital transformation through hybrid cloud adoption enables businesses to conduct operations flexibly and remotely. Consequently, businesses must persistently emphasize corporate continuity to maintain a competitive edge, particularly concerning cybersecurity. Companies need to prioritize lowering the probability of experiencing an attack, mitigate the adverse effects of an attack when it occurs, and swiftly restore normal operations after an attack.

Today, the volume and sophistication of cyberattacks has risen and will continue to increase with the use of AI, quantum computers, and other emerging technology. According to Robert Muggah and Mac Margolis, it has been estimated that cybercrime damages, including recovery and remediation, cost "\$3 trillion in 2015, \$6 trillion in 2021, and could reach \$10.5 trillion annually by 2025"¹. Therefore, you need to protect yourself now and future-proof your environment as much as possible from the attacks that are to come. It is not just external bad actors that are concerns, as inside threats can be even more damaging. The implementation of proactive protection measures, even beyond the minimum requirements dictated by your company's internal security policy or external regulation, is critical to risk reduction. Waiting to react until after there is a data breach, ransomware attack, or other security incident is too late.

For decades, IBM Z has been a leader when it comes to security. Because of the mission critical workloads that run on the platform, security is at the heart of the IBM Z and z/TPF architecture and this focus on security continues to evolve with innovative capabilities to address new cybersecurity challenges. With IBM Z and z/TPF, you can secure your business in the following ways:

Data protection now and for the future

- ▶ Encrypt all data at rest on z/TPF using quantum safe algorithms.
- ▶ Encrypt all data in flight on z/TPF using the latest industry recommended cipher algorithms with Perfect Forward Secrecy (PFS).
- ▶ Protect your data in use on IBM Z with automatic encryption in memory and mark areas of storage as nondisplayable to prevent it being viewed by operators or diagnostic tools.
- ▶ Limit access to sensitive data by configuring access control policies that restrict what data users can access or update.
- ▶ Protect your data from corruption by creating cyber resilient point-in-time copies with the IBM Safeguarded Copy function from the IBM DS8000® family of products.

Cryptographic agility

- ▶ Enable encryption or change cryptographic methodologies transparently without any application changes.
- ▶ Create cryptographic inventory reports that describe the security posture for your z/TPF environment.

¹ [Why we need global rules to crack down on cybercrime](#)

System protection and resiliency

- ▶ Protect what is loaded to your system with IBM Z® secure boot technology (IBM z16™) and the z/TPF program loader.
- ▶ Detect and prevent overload conditions with z/TPF capabilities that monitor and manage excessive traffic and resource usage.
- ▶ Take advantage of the z/TPF architecture to increase system and workload resiliency, ensuring business continuity in the event of malicious attacks or code failures.
- ▶ Use hardware redundancy to ensure there are no single points of failure.

With z/TPF and IBM Z, you have the peace of mind that your system and the data on it are secure and always protected. The z/TPF system is not security by obscurity but rather security by design.

1.1 Getting started

In this IBM Redbook, we describe key features of z/TPF security and how they align with standard security and risk management frameworks. You can utilize these frameworks as a guide to securing your business in two important ways:

- ▶ Risk management frameworks help you to integrate privacy and security into your software development life cycle, shifting security design and testing left.
- ▶ The [National Institute of Standards and Technology \(NIST\) Cybersecurity Framework](#) helps you to identify your assets and risk, build a plan to protect your business, and evaluate your ability to detect, respond to, and recover from a cybersecurity attack.

Resiliency is also a critical part of your business continuity and security strategies and is a foundational part of the z/TPF and IBM Z architecture. This includes capabilities such as high availability, workload prioritization and data isolation. We touch on aspects of resiliency throughout this Redbook, and additional details on why z/TPF and IBM Z are resilient can be found in the Appendix A, “z/TPF system and workload resiliency” on page 89.



Security by design with risk management frameworks

This chapter discusses strategies for enhancing the security of enterprise systems through effective risk management and integration of security principles across the software development life cycle (SDLC). IBM employs the Security and Privacy by Design (SPbD) framework for developing its products, including z/TPF. By adopting a risk management framework, organizations can proactively address security concerns and minimize potential cyber threats.

One way to better secure your enterprise for the future is to incorporate a risk management framework. Risk management frameworks integrate information security, privacy, and risk management activities into your software development lifecycle (SDLC) to ensure that security is in the forefront of every development project instead of an afterthought. IBM has adopted the [Security and Privacy by Design \(SPbD\) framework](#) for the development of z/TPF and other IBM products.

By shifting security and privacy to the left as early on as possible in the design and development of software, you can ensure that the security posture meets your business standards and reduces your cybersecurity risk. Some best practices for integrating security concepts into your application design include:

Ensuring applications have cryptographic agility

- ▶ Isolate your applications from the ciphers being used so that if you need to change or upgrade cryptographic ciphers, you can do so without application changes. For example, use z/TPF middleware packages to handle the Transport Layer Security (TLS) secure network configuration, so you can simply update the TLS configuration files to upgrade ciphers used to protect network flows for your application.
- ▶ Ensure you can seamlessly change cryptographic keys being used to encrypt data in the database without requiring application changes by leveraging z/TPF automatic database encryption support and the z/TPF secure keystore.

Improve resiliency using z/TPF asynchronous middleware

- ▶ Prevent one problematic remote service from impacting your entire system by using the z/TPF asynchronous model to make external calls to remote servers during transactional workloads. With the aid of an asynchronous model, the caller's Entry Control Block (ECB) can exit while waiting for the response to be sent by the remote server. This approach is superior to using synchronous calls, where the process/thread (ECB in z/TPF terminology) remains active but suspended as it waits for the response. Synchronous calls can be problematic, especially when a remote system is slow in responding because hundreds of transactional ECBS can be active waiting for responses. If this results in a low ECB condition it can impact other transactional work with longer response times or even timeouts. Asynchronous calls provide more resiliency because critical system resources (like ECBS) do not remain active while waiting for responses from a remote server, which prevents one slow responding remote server from impacting other work in the system.

Designing and creating services to limit exposure to sensitive data

- ▶ Ensure that RESTful services are designed to restrict access to personally identifiable information (PII) and other confidential data solely to those who require it for legitimate purposes. In many environments you should create separate services for your data based on who is asking for that data. For example, when an airport gate agent is accessing the airline reservation record for a passenger about to board an airline, that agent has no business need to see the details of the credit card that was used to pay for the flight. When analytics are being done on all of the new reservations that were made, that process has no business need for the payment details (for example, credit card number) or the passenger's PII.
- ▶ Limit the number of services that can expose large amounts of sensitive data. For example, in certain rare situations you might need a service to obtain all the passengers that flew into or out of a particular airport on a given day. If a service like this is required, ensure only end users with the right credentials are allowed access to the service, and as added protection, have only that service enabled/active when it is needed.

Adopting a Risk Management Framework (RMF) and integrating security into every stage of your software development lifecycle helps minimize cybersecurity risks. Additional z/TPF application design recommendations that are related to the NIST cybersecurity framework pillars can be found throughout the rest of this Redbook.



Understanding and reducing risk with the NIST Cybersecurity Framework

This chapter introduces the NIST cybersecurity framework and how it can be used to help businesses enhance their security posture.

The NIST cybersecurity framework helps businesses better understand, manage, and reduce their cybersecurity risk. By offering a structured set of recommendations, it enables businesses to safeguard their digital assets and infrastructure effectively. The NIST framework focuses on helping businesses enhance their security posture through the following major pillars:

Identify

The Identify pillar helps organizations to thoroughly understand their most important assets and resources. This includes assessing what must be protected, where there are exposures, and what is the current security posture.

Protect

The Protect pillar provides guidance on the technical security controls for protecting your critical assets. This includes access control, data security and protection now and for the future, as well as processes and procedures to ensure the protection of your data evolves as your business does.

Detect

The Detect functions alert an organization to attacks through continuous monitoring by identifying anomalies and events so that an action can be taken to respond to the attack.

Respond and Recover

Under the Respond and Recover pillars, you ensure the appropriate response to a cybersecurity event and how to recover from it. Recovering from a cybersecurity event ensures the continuity of your business in a timely manner after a cybersecurity attack or security breach.

The following sections delve into further specifics of every pillar of the NIST cybersecurity framework, highlighting the capabilities and functionalities of the z/TPF operating system while providing recommendations for integrating security aspects into software development projects.



Identifying your cybersecurity risks

This chapter describes recommendations and highlights z/TPF capabilities and functionality aligned with the Identify pillar of the NIST cybersecurity framework to help businesses understand their most important assets and resources. This includes assessing what must be protected and where there are risks of exposures, as well as determining your current security posture.

The first step in identifying what to protect is to understand what you are required to do, either by law, external regulations, internal policy, or contractual agreement with a business partner. This can identify which data, services, and systems are in scope, and what methods are acceptable like, for example, the data that must be encrypted using the AES-256 algorithm.

What you are required to protect should be considered the starting point, rather than the absolute limit. Even if you pass a security audit, you might still have significant risk exposure; therefore, security adherence and compliance should not be treated as an exercise to just check off a box on a list. The number of data privacy laws and regulations are growing at an accelerated pace where data that used to be out of scope might suddenly be considered sensitive data. Instead of expending substantial resources annually to determine what data you possess falls within the scope and which networks carry data pertinent to those scopes, adopt an "encrypt everything" strategy to safeguard both your present situation and potential future scenarios.

4.1 Threat target analysis

A key aspect to protecting yourself from cyberattacks is understanding and identifying what your threat targets are and what the business impacts would be if an attack against these targets were successful. You want to be proactive, not reactive, in your security design. Threat targets refer to the regions inside your organization susceptible to being attacked by malicious actors. Perform threat modeling to understand potential vulnerabilities or the absence of safeguards in your enterprise. This way, changes and corrections to your environment can be prioritized based on the risk assessment of the potential threats. There are many threat modeling frameworks available today to help identify your threat targets and to help quantify the threats against your business. See Appendix B, “Identify your threat targets” on page 97 for information about understanding and identifying threat targets within your enterprise.

4.2 Regulation adherence

Certain industry workloads, including finance, have historically been subject to regulation. However, due to the increasing prevalence of online access resulting from the internet and mobile technology, new regulatory frameworks are being established at a faster pace than ever before, specifying how data is allowed to be utilized and where it must be stored. The rules are becoming more stringent and are applicable to most, if not all, workloads. Some industries, like the payment card industry, are more regulated than others, but every business must deal with regulatory compliance. As the laws and the complexity of them have increased, entire business units have been created to deal with compliance and the required regulations.

For example, any business dealing with credit card processing must adhere to the Payment Card Industry Data Security Standard (PCI DSS) specification (www.pcisecuritystandards.org). With each release of a new PCI DSS specification more stringent rules on the data and the access to it are put in place. For example, the PCI DSS specification 4.0 requires customers that are using disk-level encryption, such as the DASD control unit encryption, are now required to encrypt credit card data by additional mechanisms as well. For z/TPF customers, this means encrypting credit card data within the operating system itself.

If companies fail to comply with the regulatory compliance laws, large monetary fines can be placed on the company for each violation. Proving a given regulation is adhered to by a company is often time-consuming and difficult. To illustrate, consider a scenario where a company has 200 databases distributed across its operations. In such cases, verifying whether these databases store sensitive data and ensuring proper encryption becomes crucial. Implementing widespread encryption, also known as pervasive encryption, is an effective strategy to reduce compliance burden. This approach entails encrypting every bit of data, including files at rest and in transit. The z/TPF operating system, combined with IBM Z hardware cryptography, facilitates the achievement of pervasive encryption goals, thereby minimizing the overall compliance impact.

4.3 Compliance reporting on z/TPF

Businesses prioritizing security now face greater importance in adhering to compliance regulations. Adhering to security compliance regulations helps minimize your security risk and ensure your business follows or exceeds the industry standards for security. z/TPF includes tools to assist you in determining your regulation adherence and security compliance for your networks, databases, and applications. You can use these same tools to also understand your security posture in each area and determine if you should go beyond the security compliance requirements.

Ensuring adherence to security compliance regulations involves several steps, including inspecting the network security of your z/TPF system. Understanding the network requirements for your business is crucial. As an example, your company might require enforcing secure connections using TLS 1.2 or higher. Furthermore, specific ciphers like Triple Data Encryption Algorithm (TDEA), also known as TDES, could be disallowed based on your internal security policies or industry regulations.

The ZDCOM operator command generates a report that allows you to easily understand your network security posture and can help you determine what changes need to be made to keep your systems compliant. The report shows which applications (network ports) are using secure TLS connections and which are not. The following report shows that all z/TPF applications except HTTP81 are using secure TLS network connections:

```
ZDCOM DISPLAY ALL
PORT APPLNAME PROTO TLS MODEL
-----
71 HTTPS71 TCP Y SERVER
81 HTTP81 TCP N SERVER
25050 PAYMENTS TCP Y SERVER
26020 BOOKINGS TCP Y SERVER
26030 EXSERV2 TCP Y SERVER
END OF DISPLAY
```

Simply because a network connection uses TLS, it does not automatically imply that the encryption strength of that TLS connection meets strict internal or external security standards. For instance, your policy may necessitate a connection of at least TLS 1.2 and obligate the usage of AES-128 or AES-256 for encryption of network traffic. Compliance reports for z/TPF can inform you whether your applications employ TLS 1.1 (or outdated TLS protocols) and if there is any application utilizing an older TLS protocol although it is not actively utilized yet. These reports also demonstrate if any applications utilize ciphers apart from AES-128 or AES-256, like Triple DES (TDES), or if an older cipher is permitted by an application but not being employed right now. For example, the following report indicates that the PAYMENTS application (port 25050) and EXSERV2 application (port 26030) allow the TDES cipher algorithm to be used. Nevertheless, only the EXSERV2 application features sessions employing the TDES cipher algorithm:

```
ZDCOM USAGE CIPHER-DES-CBC3-SHA
PORT MODEL CIPHER USED
-----
25050 SERVER N
26030 SERVER Y
END OF DISPLAY
```

Since the report indicates that the z/TPF application PAYMENTS is allowing an older (and now unacceptable by policy) cipher to be used, but no sessions have been established using the TDES cipher algorithm anymore, you can update the PAYMENTS application TLS configuration on z/TPF to remove the TDES cipher algorithm from the allowable list of ciphers.

Because the report indicates that one or more network connections with application EXSERV2 are using the TDES cipher algorithm, you can generate another report to show which remote users negotiated to use the TDES cipher algorithm. The following report shows the IP addresses of the five remote clients that have used or are currently using the TDES cipher algorithm to communicate with the EXSERV2 application:

REMOTE IP	LAST CONNECTION
198.51.100.20	2024-02-14 10:50:28
198.51.100.30	2024-02-10 00:37:15
203.0.113.50	2024-02-09 03:23:52
203.0.113.60	2024-02-08 16:31:31
192.0.2.70	2024-02-08 10:17:40

END OF DISPLAY

This information allows you to contact those users (other parts of your enterprise or your business partners) to upgrade their TLS configurations to use a cipher algorithm allowed by your policy.

In addition to restrictions on TLS protocols and ciphers used, your organization may enforce periodic renewal of TLS certificates, such as every ninety days. Determining which of your TLS certificates are nearing expiration can be challenging. Even worse, if an active certificate were to expire, your business could be impacted because no remote user would be able to connect to that z/TPF application until the certificate is renewed. To prevent that from happening, you can use the z/TPF compliance tooling to build a report showing you the z/TPF TLS certificates and their expiration dates. The following report shows that there are three z/TPF certificates and which application (network port) is using it. For each certificate the report also indicates when the certificate was issued, when it will expire, and how many days until that certificate expires.

PORT	MODEL	ISSUED	EXPIRES	DAYS
49020	CLIENT	Mar 08 2024	Jul 07 2024	111
26030	SERVER	Mar 13 2024	Jul 12 2024	116
26020	SERVER	Jan 09 2024	Aug 24 2024	159

END OF DISPLAY

You can also automate the process of determining which certificates are nearing expiration by configuring the TLS certificate expiration monitor user exit. For example, you can configure the TLS certificate expiration monitor user exit to alert the operator about all certificates that will expire within the next month. You can also issue a warning message for certificates that are within two weeks of expiring. Since the TLS certificate expiration monitor is driven through a user exit, the notifications and their frequency are completely up to you.

Databases are another critical area of concern when it comes to security compliance. Regulations might require that you have certain information, such as PII, encrypted in your databases with a quantum safe cipher, and during an audit you may need to prove this to an auditor. The **ZUDFM ENCRYPT USERSUM** operator command will generate a report showing which of your z/TPFD user databases are encrypted and which ones are not.

For databases that are encrypted, the report will show which encryption algorithm is being used to encrypt that database. With this report, you can easily prove to an auditor that your databases are encrypted using an approved algorithm, such as the quantum safe cipher AES-256. The following report shows that databases CSTMERS and FLIGHTS are encrypted with AES-256-CBC, and database APPCONF is not encrypted:

ZUDFM ENCRYPT USERSUM					
FILE ID	DBNAME	ENCRYPTION	KEY NAME	CIPHER	DIGEST
A270	APPCONF	N	NONE	NONE	NONE
C311	CSTMERS	Y	CSENCK1	AES256CBC	SHA256
F120	FLIGHTS	Y	FLENCK1	AES256CBC	SHA256
END OF DISPLAY					

You would then confirm that database APPCONF does not contain any sensitive information.

Networks and databases are not the only places where encryption is used. Applications can also use cryptography directly. Therefore, it is important to have a complete cryptographic inventory of your z/TPF system to prove that old and unacceptable cryptographic algorithms are not being used. The following report shows what application programs have used which secure symmetric keys, and the cipher associated with each of those keys.

ZDCOM SKEY DISPLAY SUMMARY			
PROGRAM	OPERATION	KEY NAME	CIPHER
BKSC	ENCRYPT	BKENCK1	AES256
BKSC	DECRYPT	BKDECK1	AES256
FLSC	ENCRYPT	FLENCK1	TDES
FLSC	DECRYPT	FLDECK1	TDES
END OF DISPLAY			

The benefits of maintaining a cryptographic inventory extend far beyond merely generating audit reports. There have been instances where critical vulnerabilities have been discovered in widely used TLS protocols or encryption ciphers. This poses a pressing need to identify every instance within your organization where the vulnerable algorithm is employed, followed by swift action to rectify all identified locations. The various cryptographic usage reports generated by z/TPF allow you to quickly identify if and where the affected algorithm is being used. For example, when generating these reports, you can filter them such that only places that are using that specific affected algorithm are included in the report. For quick remediation, z/TPF encryption usage was designed using configuration files and operator commands. You can change which TLS protocol and ciphers an application is using, or what cipher is being used to encrypt a database by changing the configuration. Having this cryptographic agility prevents you from making extensive changes to your application programs.



Protecting your z/TPF data and resources

This chapter describes recommendations and highlights z/TPF capabilities and functionality aligned with the Protect pillar of the NIST cybersecurity framework to provide guidance on protecting your critical assets. This includes access control, system security and data protection now and for the future.

Data that resides on your z/TPF system is one of the primary assets of your company. Therefore, protecting that data is of utmost importance. There are multiple layers when it comes to protection, including physical security of servers, network routers, and other hardware components. Protection encompasses appropriate access controls over services and data, employing encryption and various countermeasures to always safeguard data - during storage, transmission, processing as well as a tightly controlled process for loading code that involves multiple roles with clear separation of duties.

Business continuity means keeping your application services up and always running. Long ago, the focus was having redundant hardware in the primary data center and having at least one disaster recovery (DR) site. Today, the scope of business continuity must account for external and internal threats as well. The goal is to have the necessary protection mechanisms in place such that these attacks are prevented and do not impact your business.

Prevention should also be forward-looking, meaning not just aimed at thwarting the attacks of today, but also using mechanisms that will protect you from future attacks as well. A prime example of this is using quantum safe cryptography to protect your data. Protection is an ongoing process where you need to upgrade the methods used as technology evolves and incorporate protection when developing any new services or databases. There are many protection mechanisms applicable for your z/TPF environment and we cover these in more detail in the following sections.

5.1 Foundational protection

5.1.1 Maintaining currency

One of the best ways to protect your z/TPF system is to keep your hardware and software current, protecting you against known defects, even when they are not security-related. Having the system at the latest levels of firmware and software ensures that any known defect fixes have been applied to your system. This can protect you from unforeseen hardware and software errors that can cause unplanned outages, whether security related or not. For any high severity Authorized Program Analysis Report (APAR) against the z/TPF family of products, a service bulletin will be issued to communicate the severity of the problem and provide links to the fixes.

For security-related fixes, IBM continuously monitors and scans IBM products and the source code associated with them for security-related vulnerabilities. For example, the z/TPF system uses the open-source [OpenSSL library](#) for securing data in flight across the network. When a new OpenSSL vulnerability is reported by the OpenSSL community, IBM analyzes the issue to determine whether the z/TPF product is affected. If it is, a z/TPF APAR is delivered to provide the fix. In the Java space, a new z/TPF Java refresh is delivered by IBM at least every quarter that includes fixes for all security-related vulnerabilities affecting the z/TPF Java packages in that quarter. Each time a new security vulnerability is discovered by IBM for the z/TPF system, a z/TPF security bulletin is promptly released to describe the issue and the APAR required to rectify it. It is strongly encouraged that you sign up for z/TPF security bulletins and notifications regarding any other IBM products you utilize. For more information, see [IBM Product Security Central](#).

5.1.2 Securely installing software and firmware

While maintaining currency of firmware and software is important, ensuring that what is being loaded to your system is authentic and safe is paramount. IBM and the z/TPF operating system provide many safeguards to ensure protections against unauthorized program loads.

With IBM z16, quantum safe secure boot technology helps protect your firmware from attacks. This hardware-protected verification of the firmware uses a dual-signature scheme, which uses a combination of quantum safe and classical digital signatures. The IBM z16 secure boot technology provides protection through many firmware layers that are loaded during the boot process. With this approach, only authentic IBM-approved firmware is accepted, ensuring that the system starts securely by keeping unauthorized firmware (or malware) from taking over during startup. For more information, see [Prepare for the next era of computing with quantum-safe cryptography on IBMz16](#) and [IBM z16: Reasons to upgrade from your z15® or z14](#).

The vast majority of z/TPF operating system code is delivered to the customer as source code. Customers obtain the z/TPF source code directly from an IBM website using a secure channel. On other operating systems, executable code can be transferred directly to a file system using Secure File Transfer Protocol (SFTP), for example, or even programmatically through application code. This executable program can then easily be run by a user introducing malicious software into the system. Figure 5-1 on page 19 shows the inherent security vulnerabilities of loading and running malicious applications on other platforms. In this example, a user on a Linux shell transfers a Java application file onto the Linux file system and then executes that application.

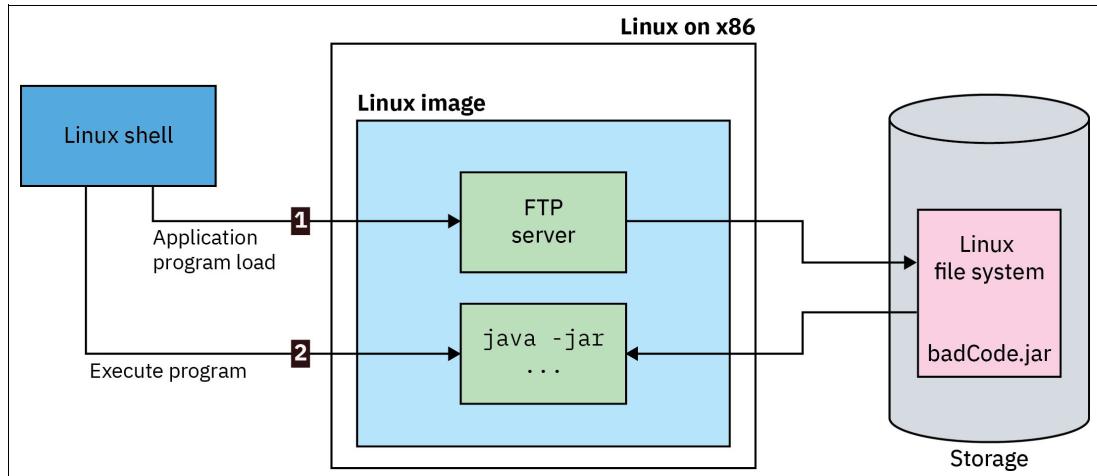


Figure 5-1 Security vulnerabilities of loading and running malicious code on other platforms

On z/TPF, executable code can only be loaded onto the system using the standard z/TPF loader. Access to the z/TPF loader can be tightly controlled allowing only a subset of authorized z/TPF operators to have the ability to load code onto the z/TPF system. This eliminates the ability for remote users to load, deploy, or execute malicious code on your z/TPF system. Assembler and C/C++ applications as well as the z/TPF operating system code do not reside in the file system on z/TPF. There is a separate area on Direct Access Storage Devices (DASDs) that is only accessible from the z/TPF program loader where the z/TPF applications and system code resides. With this approach, only the z/TPF operator can load code onto the z/TPF system, preventing bad actors from storing code on the z/TPF file system and running it. Figure 5-2 shows the z/TPF model for loading code and the inherent security of loading code on the platform.

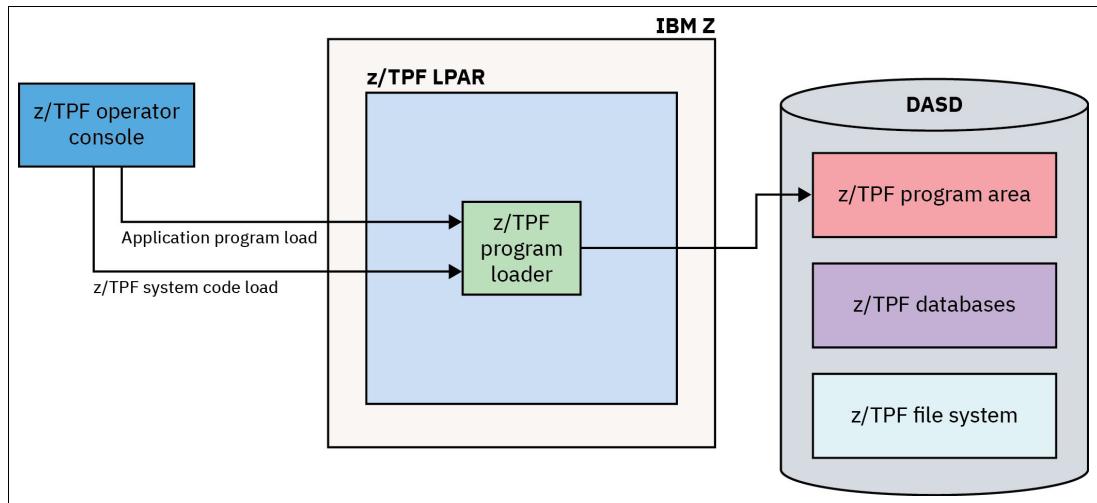


Figure 5-2 z/TPF model for loading code and the inherent security of loading code on the platform

The security and loading of a Java executable on z/TPF are also tightly controlled and can be configured to only be performed by the z/TPF operator. When loaded through the z/TPF program loader, the Java executables are stored in an area that only certain parts of the z/TPF kernel are allowed to update, like the z/TPF program loader. Loading a Java executable into the standard z/TPF file system available to applications can be prohibited on z/TPF. Figure 5-3 on page 20 illustrates the robust security measures that can be implemented when loading Java executables on the z/TPF system.

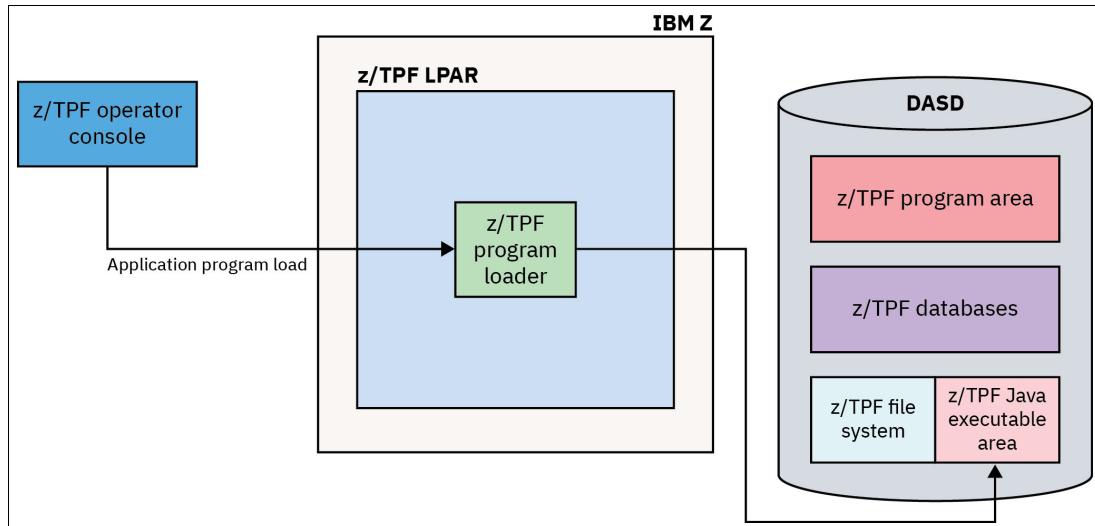


Figure 5-3 Security of loading Java executables on z/TPF

Another aspect of Java security on z/TPF is the separation between the application and the z/TPF database that may be accessed by that application. z/TPF Java programs do not have direct access to your existing z/TPF databases, whether they are older-style traditional z/TPF databases accessed through find and file APIs or the newer z/TPF Database Facility (z/TPFDF) databases. On many other platforms, executable code and databases are on the file system together and normally accessible through standard Java interfaces such as Java Database Connectivity (JDBC) drivers. In this architecture, if a defect or malicious user allows a program to obtain root access, that program might access (and update) everything on that platform including the user and system databases.

z/TPF databases are not in the Hierarchical File System (HFS). Therefore, even if a Java program managed to obtain root access in the file system, that program would not have access to your z/TPF databases. Java code can only access z/TPF databases through code (in C/C++ and assembly language) that you define as a callable service by Java code on z/TPF. For example, if the only such service you make available to Java is to read a hotel reservation record with the person's name and check in date as input, that is the only data in any of your z/TPF databases that Java code on z/TPF can obtain. For multi-language z/TPF applications, if you have code in C/C++ or assembly language that is calling a Java program, you can pass data that way as well, and in that case your application code in C/C++ or assembly language controls what data is visible to Java programs. Figure 5-4 on page 21 shows the separation between the running Java code and the z/TPF databases that Java code may be accessing.

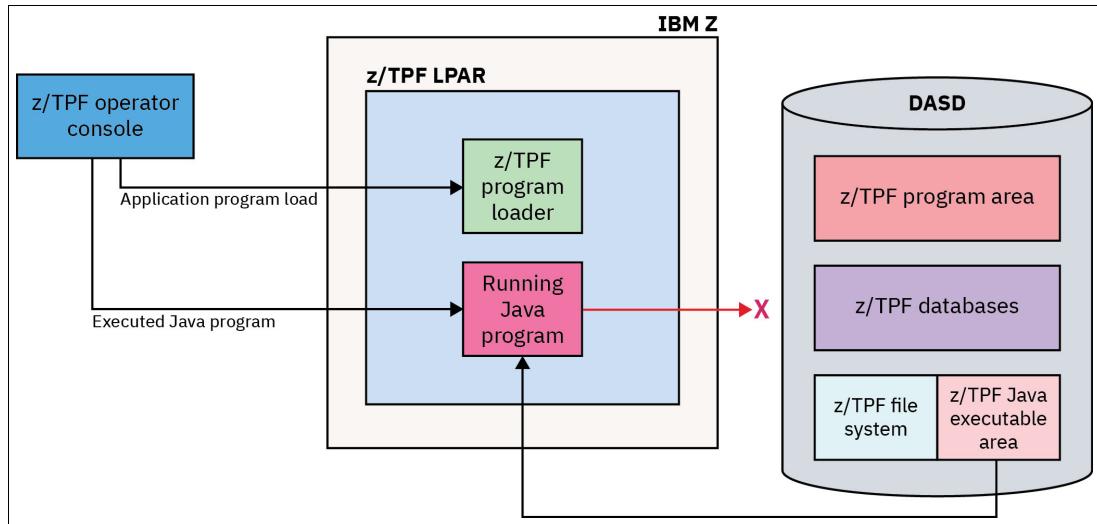


Figure 5-4 Separation between running Java code and z/TPF databases

5.1.3 Future proofing your protection

When protecting your data, you do not want to solely focus on protection for today's world. You want to keep in mind protection for the future. When it comes to security, one thing is certain, the changing security landscape over time is expected and is occurring quickly. There are aspects of protecting your data today that will enable agility and better protection for the future.

One way to position your data protection posture for the future is to ensure you have cryptographic agility so you can quickly respond to changing regulations and cryptographic algorithms. You can achieve cryptographic agility in different ways. One way to do this is to separate your application from the ciphers being used. With cryptographic agility, you can easily identify and change a compromised cipher without having to change application programs or require any downtime.

Some ways you can obtain cryptographic agility on your z/TPF system include the following:

- ▶ Avoid embedding cryptographic configuration such as ciphers, Transport Layer Security (TLS) protocols, keys, or certificates directly into the application. Design your applications in a way where changing the cryptographic configuration is easy to do, for example, by using z/TPF TLS configuration files for all your applications and middleware.
- ▶ Whenever possible, use z/TPF middleware for networking traffic, like the Internet Daemon (INETD) to control your secure network applications. This way, changing ciphers or TLS protocols is as simple as updating the configuration for that application. For example, say you have an INETD server that is using AES-128-CBC ciphers. In the future, it is identified that the AES-128 ciphers are no longer acceptable by your business or have been compromised in the industry. By having the definitions in a common INETD TLS configuration file, it is quick and easy to change the TLS configuration for an application.
- ▶ When encrypting data at rest, be sure to use the z/TPFD automatic database encryption support or the secure key APIs. z/TPFD encryption and the secure key APIs are based on the concepts of the z/TPF secure keystore. This inherently provides cryptographic agility by changing encryption keys and ciphers without having to take an outage, all while being completely transparent to applications. For more information, see “The z/TPF keystore” on page 22 and “z/TPFD database encryption” on page 42.

Another thing you can do to protect yourself for the future is by using quantum safe ciphers now. The 256-bit Advanced Encryption Standard (AES) cipher is considered quantum safe. You should encrypt your data at rest and in flight across the network with a quantum safe cipher like 256-bit AES. The industry is working towards standardizing a quantum safe solution for the key exchanged to start a TLS session. Until quantum safe key exchange for TLS becomes available, the best practice is to use ephemeral TLS ciphers that ensure PFS, in which every new TLS session creates an ephemeral public/private key pair used only for that session. The alternative to these ephemeral ciphers is the same public/private key pair is used for all your sessions. With PFS, if your private key is compromised, only the information flowing on that single session is exposed. With a shared key (non-ephemeral ciphers), the (sensitive) data across all your sessions is exposed if the private key is compromised.

Another way to future proof your environment is to “encrypt everything.” There is more than enough cryptography capacity on IBM Z to encrypt all your data at rest and across the network. With an “encrypt everything” mindset, you are protected even when regulations change and mandates are put in place to require other data on your system to be encrypted. Figure 5-5 shows the ideal environment with all your data and network connections encrypted.

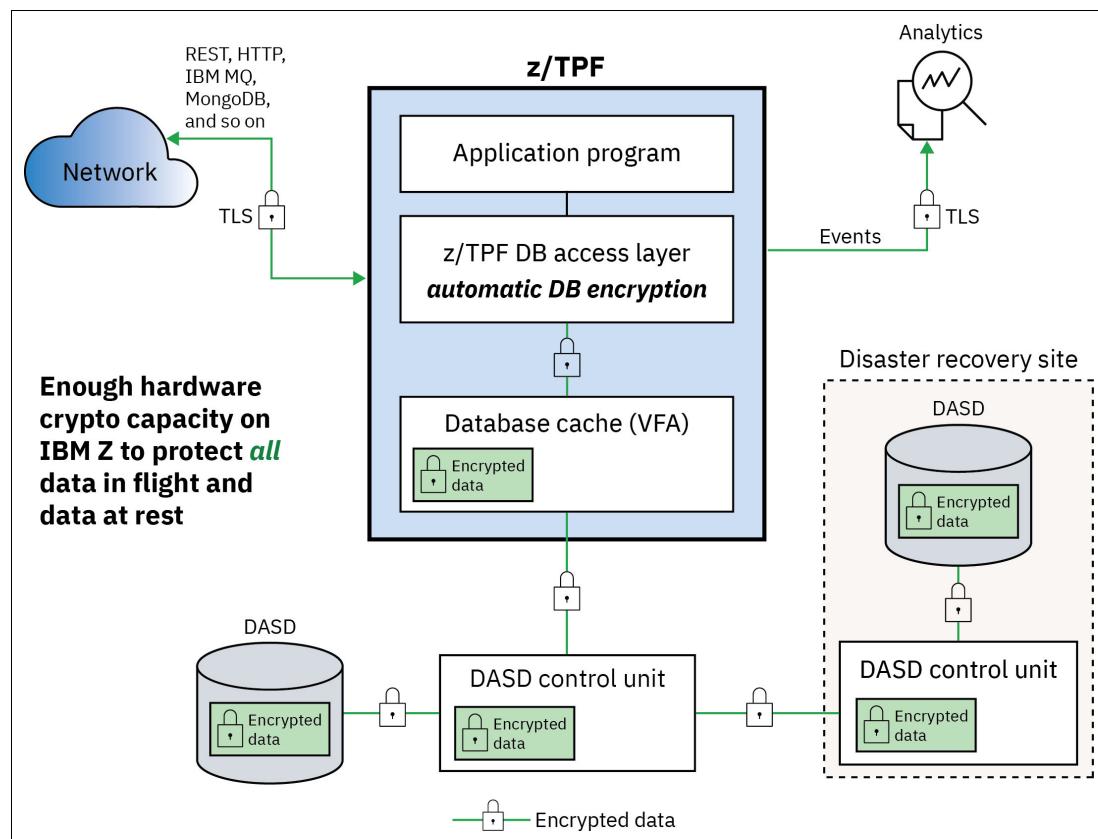


Figure 5-5 z/TPF pervasive encryption support

5.1.4 The z/TPF keystore

The z/TPF system has a secure keystore to create, import, and manage the cryptographic keys that can be used by z/TPF applications as well as z/TPF system code. The z/TPF keystore consists of a public key infrastructure (PKI) keystore and a symmetric key keystore. The public key infrastructure (PKI) keystore is used for asymmetric cryptography, for example, Rivest-Shamir-Adleman (RSA) operations that are performed when starting TLS sessions.

The symmetric key keystore is used for bulk data encryption. For example, encrypting database entries using strong cryptography like 256-bit AES, or AES-256. Keys in the keystore are referenced by name and the key values are never exposed to an application program, operator, or in diagnostic data including system dumps.

Applications as well as z/TPF system code can issue local function calls to interface with the z/TPF secure key support code that accesses the z/TPF keystore and leverages hardware crypto to encrypt and decrypt data. For example, the z/TPF TLS stack based on the OpenSSL package has hooks to call z/TPF secure key support to access a given public/private key pair in the PKI keystore to perform certificate validation when a TLS session is started. The z/TPFD package also has hooks to call z/TPF secure key support to access a given symmetric key in the symmetric key keystore to encrypt z/TPFD database records for databases defined as requiring encryption. A set of operator commands is available that allows you to manage the keys in the keystore and display information about them. Using z/TPF operator commands, you can create new keys including rotating keys transparent to application workloads. The key rotation process can also be used to upgrade to a strong cryptographic algorithm. For example, your current key is AES-128 based, and the new key is AES-256 based. While there are separate z/TPF operator commands to manage and display PKI keys and the symmetric keys on your system, there are common utilities to back up and restore the z/TPF keystore that includes both PKI and symmetric key keystores.

Figure 5-6 shows the component level view of the z/TPF keystore for a z/TPF logical partition (LPAR) running on IBM Z.

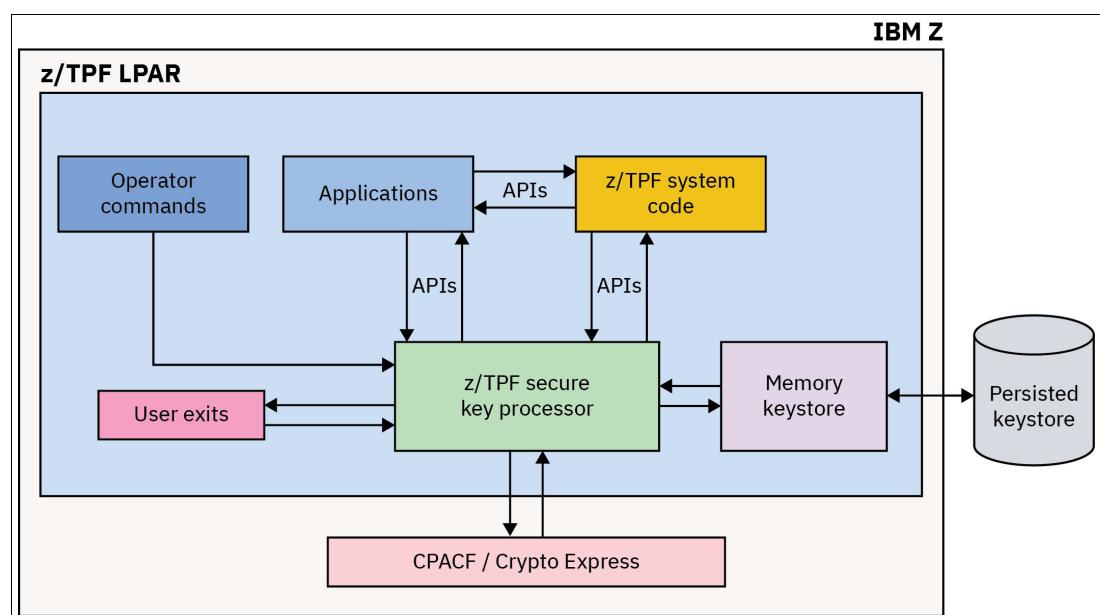


Figure 5-6 z/TPF keystore - component view

The z/TPF keystore uses the cryptographic hardware of IBM Z to accelerate cryptographic operations. The Crypto Express cards are used to perform RSA operations, which are used for certificate validation during TLS session establishment. The CP Assist for Cryptographic Functions (CPACF) is an accelerator on every IBM Z CPU that performs encryption algorithms like AES-256.

There is a key usage user exit that allows users to write their own code to verify that a given application program has access to a specific key in the keystore. The user exit allows users to centrally control access to the keys within the z/TPF keystore. The user exit also provides the ability for the z/TPF system to log user access to the keys within the keystore. This way you can build an audit log that in many cases is required for compliance. Many z/TPF workloads do hundreds of thousands of encrypt and decrypt operations per second. Therefore, it is recommended that you do not log every usage of a key as attempting to do so will overwhelm most logging mechanisms. Instead, only log unauthorized attempts at using a key.

Access to the z/TPF keystore

The z/TPF keystore persists on file (DASD) as it is a shared keystore across all the processors in a loosely coupled complex. However, a copy of the keystore is also kept in memory on each processor in a loosely coupled complex. This accelerates performance and eliminates the need for I/O to access keys during transactional workloads that are doing hundreds of thousands of cryptographic operations per second. The copy of the keystore in memory is in an area of memory that is only accessible by a small component of the operating system. In other words, almost all the z/TPF operating system code and all application code cannot see or access the memory where the memory keystore resides.

Figure 5-7 depicts the transactional view for memory access. Each transaction is processed by its own ECB. A given z/TPF ECB has read/write access to its own private area (memory) consisting of things like ECB heap, call stack areas, and core work blocks. A given ECB may also have read/write access to user tables which are shared memory tables that all transactions (ECBs) may need to read or update. For example, a user table may consist of the flight schedules for a given time. The transactional ECBs have read-only access to certain system tables, but do not have authority to update system memory. There is no access to the z/TPF memory keystore from any ECBs in the system.

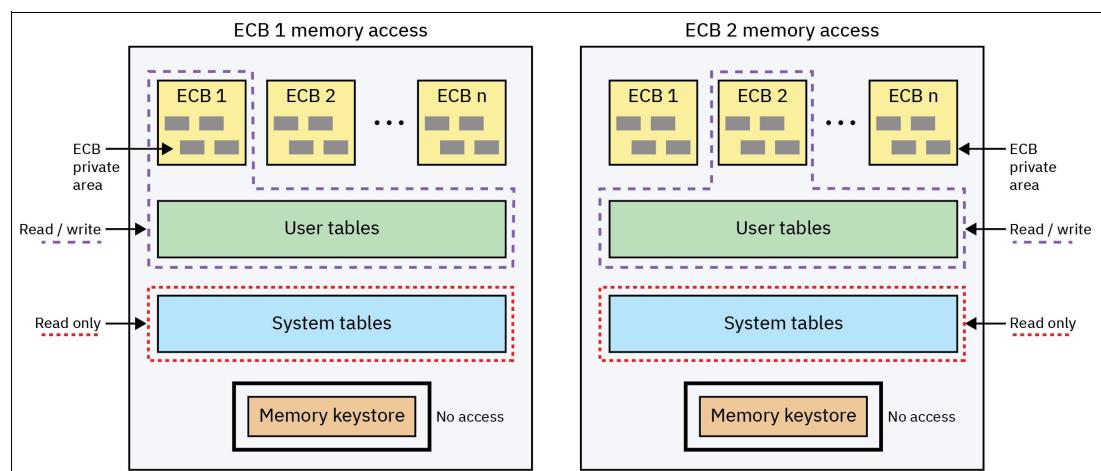


Figure 5-7 z/TPF memory ECB transaction view

Figure 5-8 shows the memory access model for the z/TPF control program. The z/TPF control program has read/write access to all memory including ECB private areas, user tables, and system tables. However, almost all the z/TPF control program cannot access the z/TPF memory keystore. Only one small module within the z/TPF control program can read or update the z/TPF memory keystore. This tight control of visibility and access to the z/TPF memory keystore provides added security, but also know for even higher security, the keys in the memory keystore are not stored in the clear in that memory.

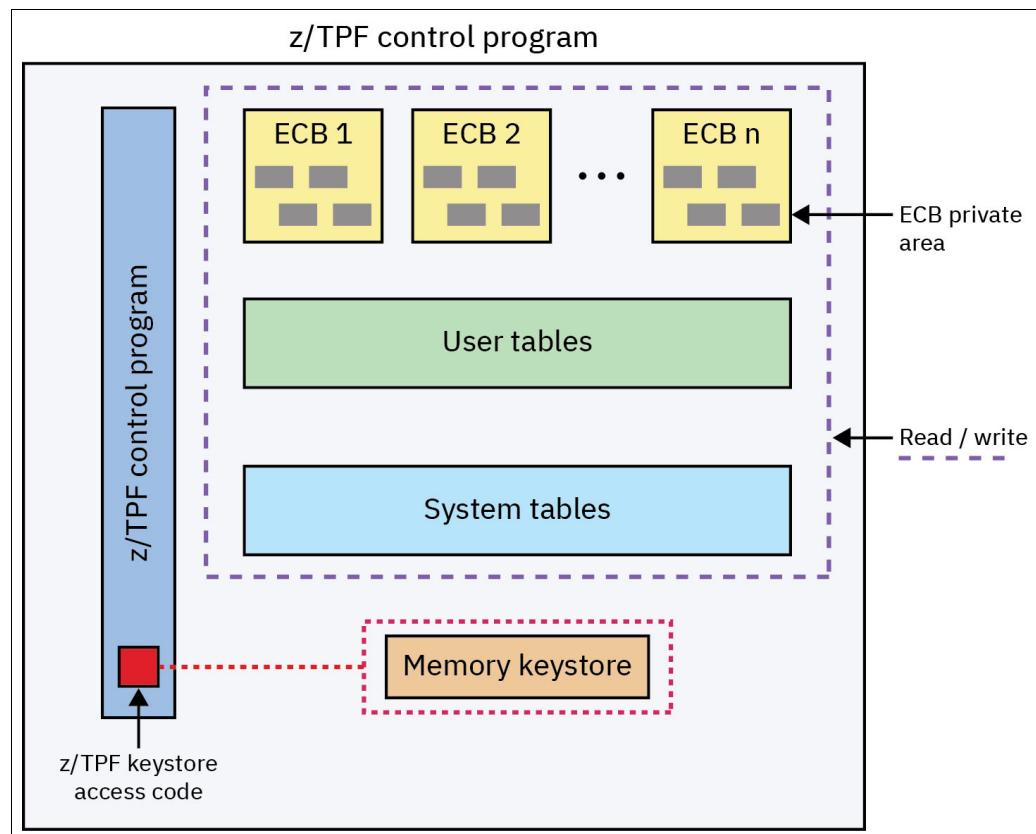


Figure 5-8 z/TPF control program view of memory

The z/TPF keystore integrated into the z/TPF system

The benefit of having the z/TPF secure keystore as part of the z/TPF system is that as the system moves, so does the keystore. For example, say you have a z/TPF system running in a data center on the west coast of the United States. Through DASD replication, that system is replicated to a data center in the east coast acting as a DR environment. Anything stored on the western data center DASD that has been updated will be replicated onto the eastern data center DASD. This includes updates to application programs, operating system code, user databases, system databases, and the keystore. Figure 5-9 on page 26 shows this replication scheme with a deactivated IBM Z processor in case of a failure of the primary data center.

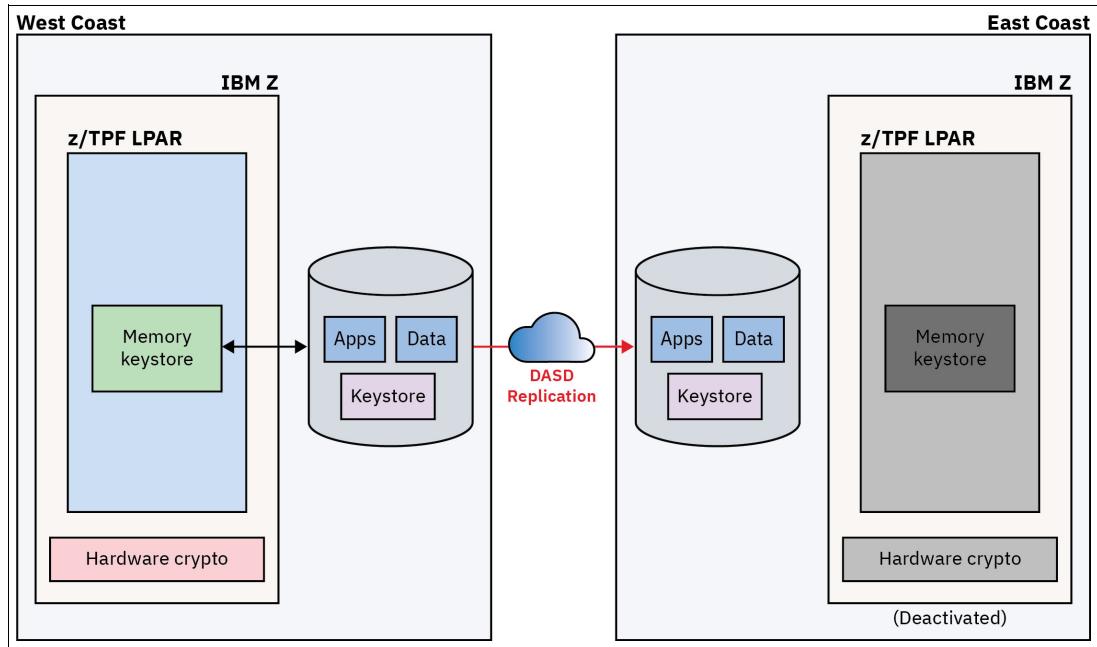


Figure 5-9 Replicating data to a disaster recovery environment

As shown in Figure 5-10, if an event occurs that makes the west coast data center unusable, you will bring up your z/TPF environment in the east coast data center. In this case, applications can continue to read and update user databases that contain encrypted data because the keystore data is replicated along with the user databases.

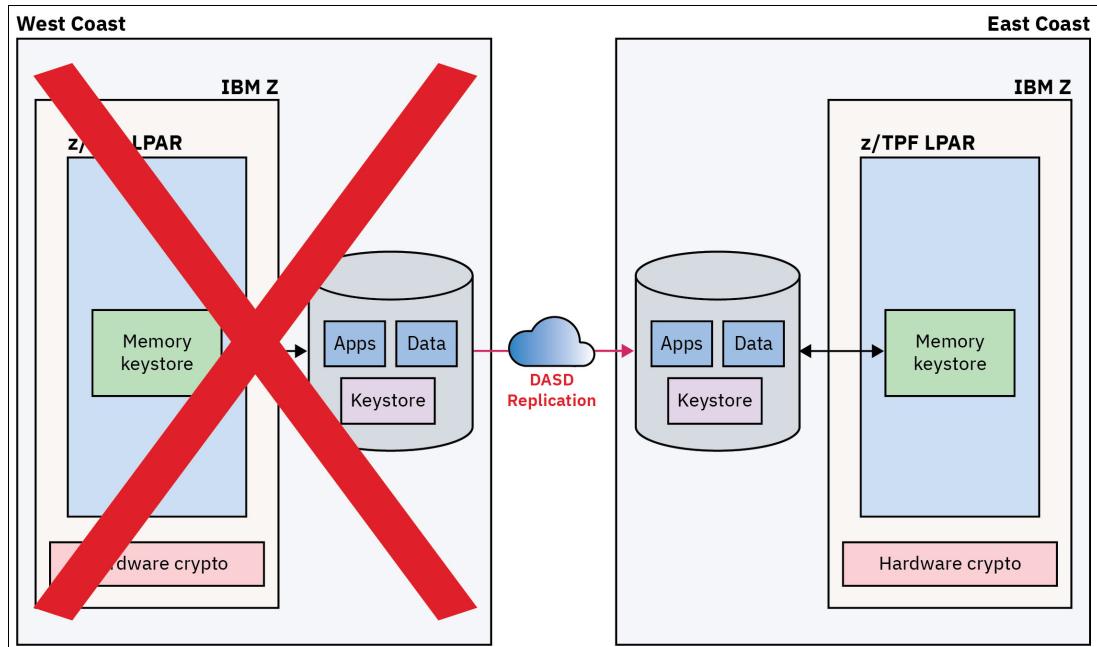


Figure 5-10 Failure of the primary data center

The z/TPF keystore and test environments

It is common practice for clients to make a copy of their z/TPF production system to refresh the z/TPF test environments. As part of this process, the data in the newly created test environment needs to be "scrubbed", meaning any PII and sensitive information like credit card numbers must be removed or changed in the test environment. When dealing with encrypted data using the keystore, you do not want the cryptographic keys in production to be used in a test environment. Migration of encrypted data using production keys to a newly created 'test' key is simple and can be combined with your existing process that scrubs PII data. For example, Figure 5-11 shows a production system encrypting a customer record using cryptographic keys from the production keystore.

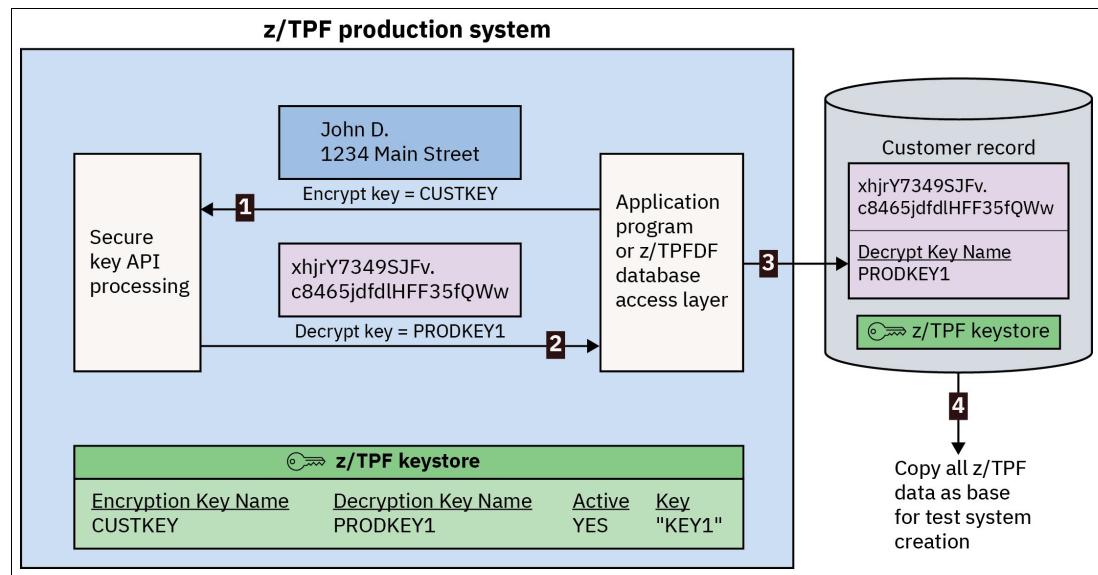


Figure 5-11 Production system encrypting a customer record using cryptographic keys from the production keystore

The steps shown in Figure 5-11 are as follows:

1. An application program on the production z/TPF system is issuing a `tpf_encrypt_data` API passing `CUSTKEY` for the encryption key name.
2. The z/TPF secure key processing encrypts the data using the production key and returns the decryption key name of `PRODKEY1`.
3. The application program saves the decryption key name along with the encrypted data.
4. At this point, the production database is copied to a different set of DASDs to be used as a test system. There could be thousands of encrypted customer records on the production system.

The production cryptographic keys must be changed to test keys to avoid using production keys in a test environment. z/TPF customers have utilities that walk through the production data in a test environment modifying (scrubbing) the sensitive data and filing the data back down to disk. Before the sensitive data is scrubbed, you must create new test keys for your databases.

Figure 5-12 on page 28 shows the creation of a new cryptographic key on the test system. Subsequently, utilities to scrub the data for individual customer records can be run.

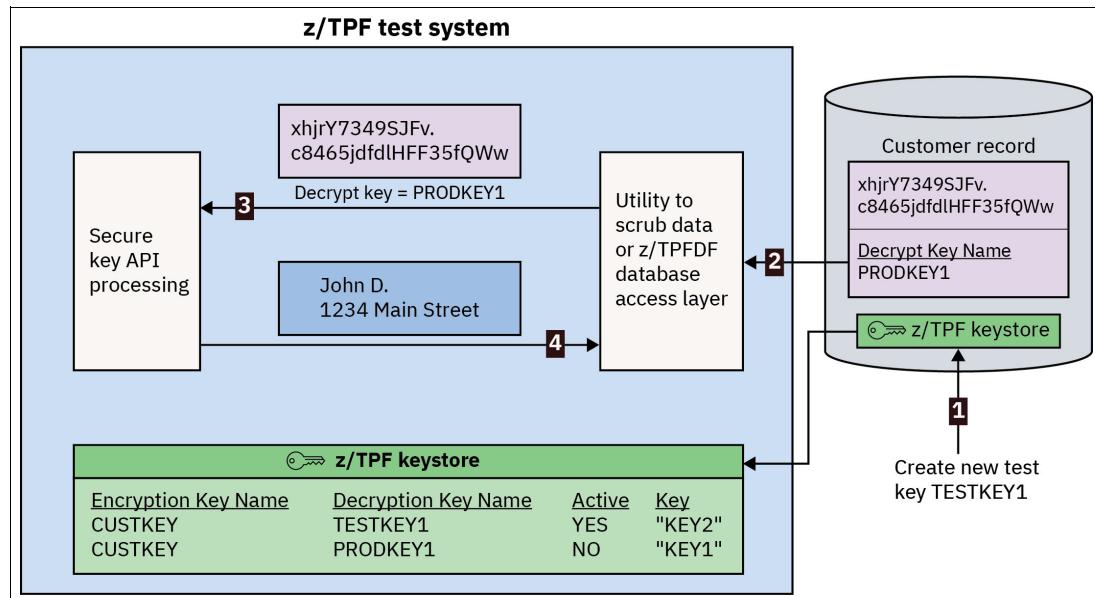


Figure 5-12 Creating a new cryptographic key on a test system

The steps shown in Figure 5-12 are as follows:

1. A new key is created for use by the test system with the same encryption key name of CUSTKEY with a different decryption key name of TESTKEY1.
2. The customer records are read from the database.
3. The tpf_decrypt_data API is issued passing the decryption key name PRODKEY1 to decrypt the customer record.
4. The decrypted data is then passed back to the application or z/TPFDF database utility.

Now that the data is decrypted, the data can be scrubbed of PII data (personal information is modified) and re-encrypted using the cryptographic key created for the test environment. Figure 5-13 on page 29 shows the customer records being re-encrypted, passing the same CUSTKEY encryption key name, but now the key created for the test environment with the decryption key name of TESTKEY1 is used.

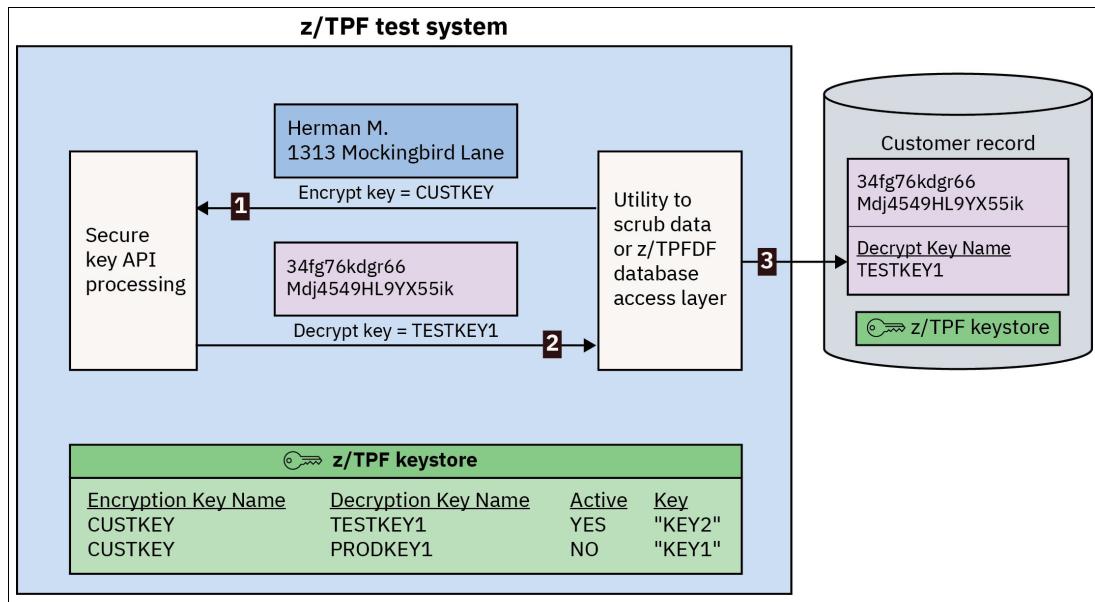


Figure 5-13 Re-encryption of customer records on a test system

The steps shown in Figure 5-13 are as follows:

1. An application program is issuing a `tpf_encrypt_data` API passing `CUSTKEY` for the encryption key name.
2. The z/TPF secure key processing encrypts the data using the test key and returns the decryption key name of `TESTKEY1`.
3. The application program saves the decryption key name along with the encrypted data.

Once the scrubbing utility completes and all data has been transitioned to use the cryptographic key created for the test environment, the production keys can now be deleted as shown in Figure 5-14.

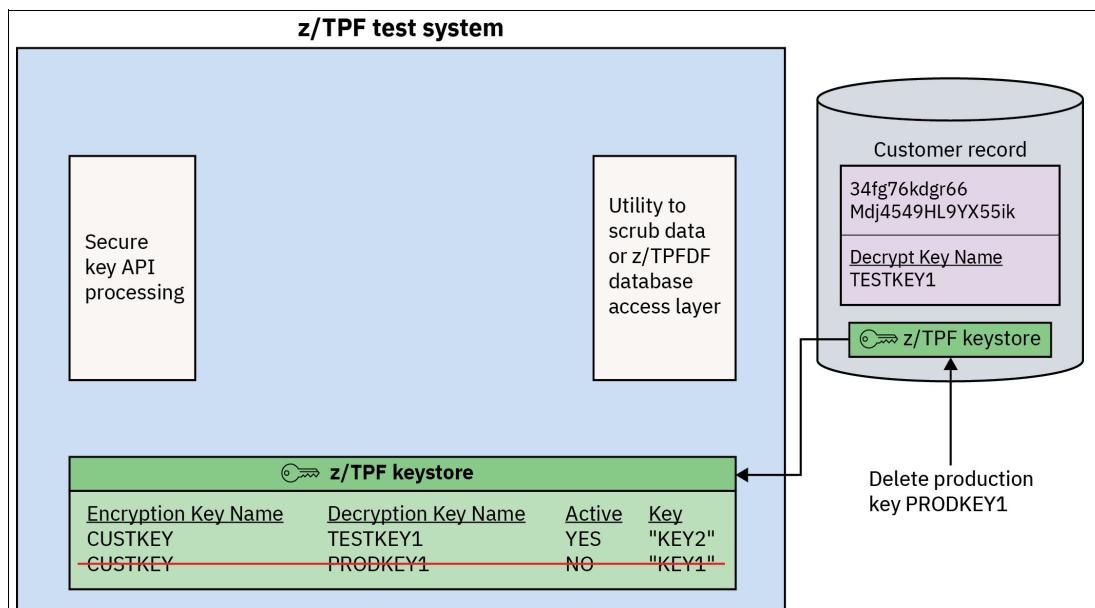


Figure 5-14 Deleting production keys from the test system

At this point the test system can be safely turned over to your systems and development labs for use. All sensitive data has been removed or changed and all production system cryptographic keys have been deleted from the test system.

5.2 Data protection

5.2.1 Securing data in flight

Securing data in flight is crucial for data protection, especially when regulations or business needs require the handling of sensitive data. z/TPF uses TLS, a cryptographic protocol used to secure data transfer between two endpoints to secure data in flight, and OpenSSL for all TLS communication to secure data in flight. The OpenSSL package on z/TPF is enhanced to use hardware cryptography on IBM Z to improve performance and was extended to allow a TLS session to be shared by multiple processes to enable long-lived TLS sessions, each of which can be used for millions of transactions over time.

z/TPF provides TLS for all its available middleware packages such as HTTP and IBM MQ. Applications on z/TPF can create, use, and manage network connections themselves, either plain (unsecure) TCP/IP sockets or TLS sessions, but recommended practice is for applications to use a standard middleware package to communicate with external users and applications, where the middleware package handles all the network connections. Regardless of who creates and manages network connections, it is highly recommended to use secure (TLS) sessions.

INETD is a system service that provides networking to user server applications and to z/TPF middleware servers. INETD can create and manage network connections, both unsecure TCP/IP sockets and secure TLS sessions. It is recommended to use INETD rather than trying to manage network connections in your server applications.

Starting a TLS session requires several pieces of information, such as which TLS protocols and ciphers are acceptable to use, the location of your private key and certificate, and so on. To centralize the configuration information of your TLS sessions, z/TPF has configuration files for you to define all the TLS properties for a given application. This allows you to easily make updates through the files, no application code changes required, as new versions of TLS are supported and as your company security policy changes.

Figure 5-15 on page 32 shows the z/TPF application networking view, where applications can interface with z/TPF middleware and INETD, communicating securely through either TLS over TCP/IP or solely through TCP/IP. Applications seamlessly communicate with the lower layers of the diagram, and from the application's perspective all data is seen in plaintext. The data sent and received is encrypted and decrypted at the TLS layer, and then sent over the TCP/IP layer. This process occurs whether your data is sent straight from the application layer, z/TPF middleware, or INETD.

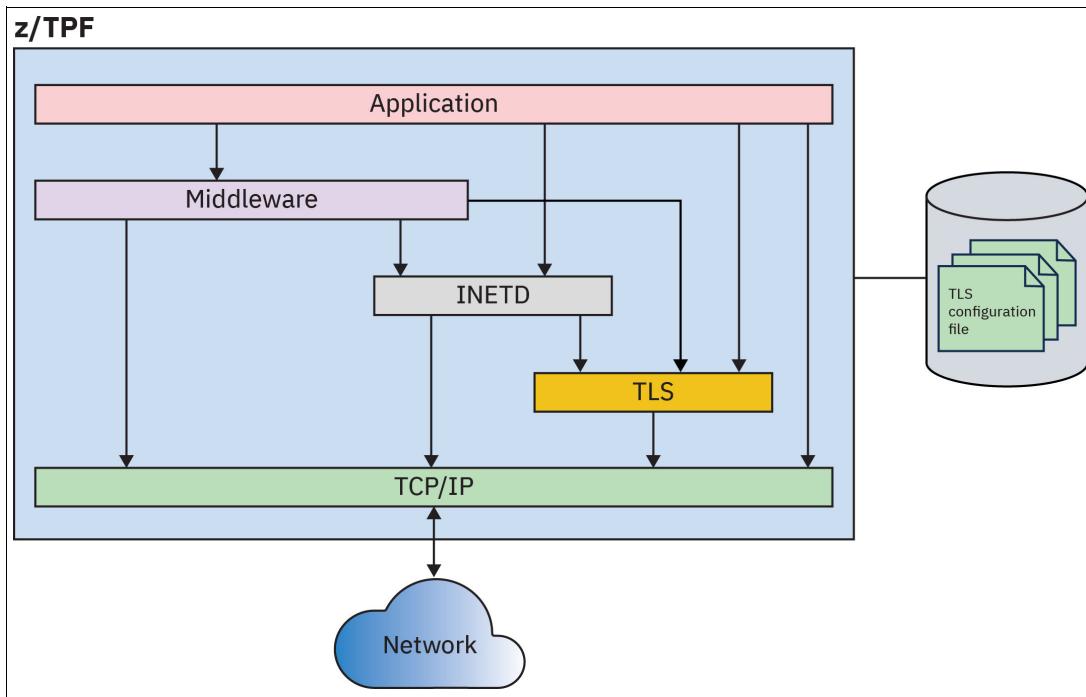


Figure 5-15 z/TPF application networking view

If you have a server application named BOOKING and you want INETD to manage its network connections and to use TLS sessions, specify MODEL-SSL when defining the server to INETD, then create the following configuration file that defines the TLS properties as illustrated in Figure 5-16.

```

INETD TLS configuration file for server BOOKING
File location: /etc/ssl/inetd/BOOKING.conf

VERSION=TLSv1_1
CIPHER=ECDHE-RSA-AES256-GCM-SHA384,ECDHE-RSA-AES128-GCM-SHA256,AES256-SHA
VERIFYPEER=YES
CERTIFICATE=/certs/booking_cert.pem
KEY=/tpfpubk/booking.pem
...

```

Figure 5-16 Example INETD TLS configuration file

TLS configuration changes using cryptographic agility

Say your organization decides that TLS 1.1 is no longer acceptable to use. To update the TLS configuration for server application BOOKING, remove TLS 1.1 as an allowed version. No application changes are required. You would do that for all TLS configuration files that currently allow TLS 1.1. You can use the network compliance tooling to identify all the applications that allow TLS 1.1. For more information, see 4.3, “Compliance reporting on z/TPF” on page 13.

Other z/TPF middleware packages follow a similar convention for defining TLS configuration files. For example, TLS configuration files for secure high-speed connector groups are in the /etc/ssl/conn directory and have the same syntax. Consistency of syntax and location across all middleware packages on z/TPF for configuring TLS sessions eases setup and maintenance of TLS configuration.

It is strongly recommended to use z/TPF middleware packages for securing data in flight given their maintainable and centralized nature. However, if you choose to manage TLS sessions within your applications, it is recommended to use and follow the same TLS configuration file standards used by the z/TPF middleware. z/TPF provides the tpf_ssl_getConfig API which parses TLS configuration files and uses the same syntax across all z/TPF middleware. Using this API and maintaining TLS configuration files in your own /etc/ssl/appname directory keeps TLS configuration out of code. It also provides the added benefit of having all of your TLS configuration for your z/TPF system in a common location on the file system (/etc/ssl).

Having TLS configuration in files instead of embedded in your application code allows you to quickly and easily react to changes in your security policy that require changes to TLS session properties. Instead of finding and changing all TLS settings embedded in application code, you only need to change the settings in the TLS configuration files. Since the TLS configuration files are separate from z/TPF middleware and your own TLS application processes, you do not need to take down the entire application process to apply configuration changes. The TLS configuration changes are updated after restarting your server, but existing TLS connections remain active.

Cipher and algorithm recommendations

One component of customizing your TLS configuration files is defining which TLS ciphers are acceptable for use by this application. A cipher contains information on how to secure each session. Up to and including TLS 1.2, the first part of the TLS cipher is the handshake algorithm, which describes how the secret symmetric key will be exchanged between the client and server. z/TPF supports both Ephemeral Diffie-Hellman (DHE) and Elliptic Curve Ephemeral Diffie-Hellman (ECDHE) key exchange methods. These ephemeral key exchange methods enforce PFS, which gives the benefit of generating a new public/private key pair for every session that is used to exchange the secret symmetric key created for that one session only. This means that if one "key exchange" key is compromised, only the data exchanged for that TLS session is compromised.

Another part of the TLS cipher is the symmetric encryption algorithm, which describes how the data communicated during a TLS session is secured in flight. z/TPF supports ciphers that use AES-256 for symmetric encryption, which is a strong symmetric encryption algorithm. The TLS cipher also specifies how the encrypted data is protected against accidental or malicious tampering by using a message digest hashing algorithm.

With TLS 1.3 the cipher suites are a simplified set of ciphers that do not contain the handshake algorithm. TLS 1.3 enforces PFS by requiring the use of ephemeral key exchange methods described previously. If you use TLS 1.2 or older, use ephemeral ciphers that have PFS to make your migration to TLS 1.3 easier.

It is recommended to use strong symmetric encryption algorithms for your ciphers and cipher suites, and key exchange methods that provide PFS for your ciphers when setting up your TLS connections on z/TPF.

Server cipher preference

As part of the TLS session establishment, a TLS server must determine which cipher to use given which ciphers the client and server support. Both TLS servers and clients can be configured to have their cipher lists of choice. These cipher lists are in preference order, so ciphers earlier in the list are preferred over ciphers later in the list.

By default when using OpenSSL, the client's preference order is taken over the server's preference order. This means that the first cipher in the client's cipher list that is also supported by the server application will be chosen and used for the specific TLS session. In Figure 5-17, the client sends its cipher list to the server in the `CLIENT_HELLO` message. The server then chooses the first matching cipher, using the client's cipher list preference. The TLS session will be established using the less-secure DES-CBC3-SHA (Triple-DES) cipher because this cipher is preferred by the client and exists in both the client's and the server's lists.

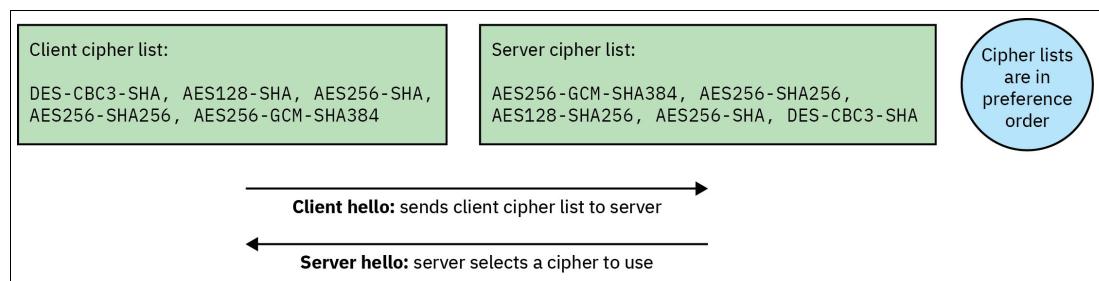


Figure 5-17 TLS cipher preference example

There is an option to change the settings of all z/TPF TLS servers, so that your z/TPF servers will take the server's cipher list preference instead of the client's. To enable this option, the keypoint 2 value `SSLSERPV` must be assigned YES either by loading a new keypoint or by issuing the `ZNKEY SSLSERPV-YES` command. With the option enabled, the prior example will now start searching through the server cipher list to find a match between both lists. Since the more secure cipher `AES-256-GCM-SHA384` matches both lists and is the first preferred server cipher, it will be chosen and used to establish the TLS session. It is recommended that you enable the option on z/TPF to use the server's preference because many clients use the default cipher list in place on the client platform, which might not be ordered with the strongest ciphers first in the list.

Server and client authentication

Another critical part of establishing a TLS session is the authentication step. This step is always performed by the TLS clients to authenticate the TLS server they are communicating with. Authentication can also optionally be done by the TLS server to authenticate the TLS client. Figure 5-18 on page 35 shows the full TLS handshake with server and client authentication for TLS 1.2 and lower, and TLS 1.3. One performance improvement with TLS 1.3 is that the TLS handshake is reduced from 4 flows down to 3. The TLS server always authenticates itself to the client by sending its certificate to the client shown in red.

When client authentication is used, the TLS client authenticates itself by sending its certificate to the server shown in blue. For business-to-consumer (B2C) connections, server authentication is typically used. For business-to-business (B2B) connections, it is recommended to use mutual authentication.

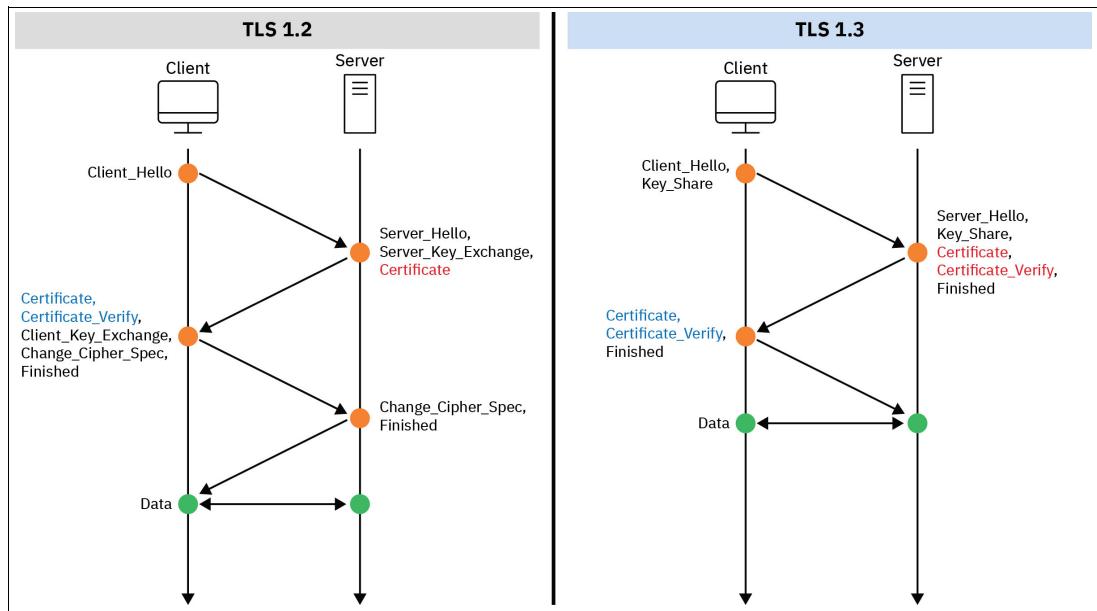


Figure 5-18 TLS handshake with client authentication

Client authentication can be enabled on z/TPF TLS servers by setting the VERIFYPEER option to YES in the TLS configuration file.

When you are configuring your z/TPF TLS client applications to communicate with TLS servers that require client authentication, your z/TPF client application must have a certificate defined because client authentication requires that the TLS client application send its certificate to the TLS server to be validated.

Hardware cryptographic acceleration

The z/TPF system offers hardware cryptographic acceleration for several portions of the TLS handshake and for communicating secure data. The IBM Z Crypto Express card is used on z/TPF for authenticating client and server certificates, and for older TLS ciphers that use RSA for key exchange. The IBM Z CPACF is hardware that can be used to perform encryption and decryption operations of data flowing across a TLS session, to generate and validate message digests used for message authentication, and to perform ECDHE key exchange. With hardware installed, z/TPF will automatically use the hardware's capabilities to speed up the previously mentioned cryptographic functionality for your TLS sessions.

Figure 5-19 on page 36 shows two different system configurations. Each z/TPF LPAR has a CPACF for each configured I-stream on that LPAR. Each IBM Z system can optionally have Crypto Express cards that can be shared by multiple LPARs. It is best practice to have connectivity to multiple Crypto Express adapters for high availability. The configuration on the left in Figure 5-19 on page 36 shares adapters Crypto Express 1 and Crypto Express 2 between LPARs TPFP1 and TPFP2. Each LPAR in the configuration on the right in Figure 5-19 on page 36 has its own dedicated Crypto Express cards, depicted by Crypto Express 3 and Crypto Express 4 for TPFP3 and Crypto Express 5 and Crypto Express 6 for TPFP4.

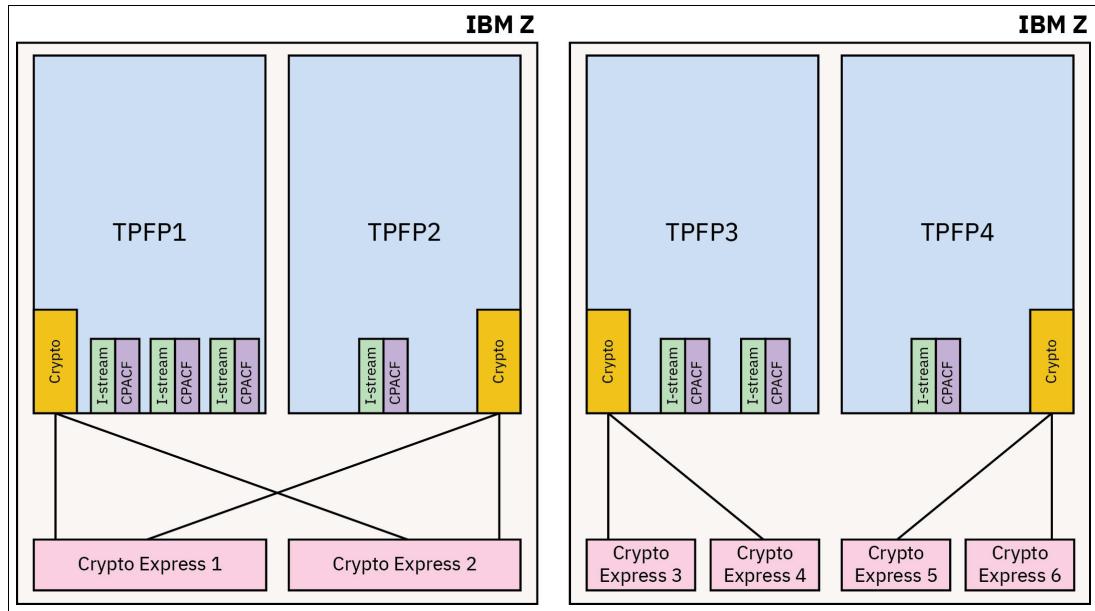


Figure 5-19 Hardware cryptography acceleration - different system configurations

The TLS handshake is an expensive set of operations, both in terms of CPU consumption and network latency. Hardware cryptographic acceleration on IBM Z processors reduces the CPU overhead, but best practice is to use long running TLS sessions whenever possible to minimize the number of TLS handshakes performed.

Shared SSL support

On other platforms, the process that creates a TLS session owns that TLS session. A given process on a distributed platform cannot access the TLS session established by another process. In z/TPF, processes (or ECBs) are constantly being created and torn down for each transaction that z/TPF processes. Given the transaction model of z/TPF, having to establish a new TLS session for each transaction processed would not be possible.

Shared SSL support in z/TPF allows TLS sessions to remain active for long periods of time without being tied to a particular process or ECB. Multiple ECBs can share one TLS session, and a TLS session can be passed from one ECB to another. Shared SSL support on z/TPF has several benefits over traditional TLS sessions:

- ▶ Multiple z/TPF transactions can flow in one long running shared SSL session even when each transaction is in its own process, reducing the number of TLS handshakes required to communicate with one remote partner.
- ▶ Allows for asynchronous I/O by issuing the z/TPF specific SSL_aor API on your shared SSL session. This allows your application ECB to exit and free up system resources while waiting for data to arrive from the remote partner. A new ECB will be created to process the work for that shared SSL session once it arrives.
- ▶ Since shared TLS sessions are independent of their creating ECB, you can have thousands of active TLS sessions with minimal active ECBs. This saves system resources over using traditional TLS sessions.

Shared SSL works by assigning the underlying OpenSSL contexts and sessions to shared SSL daemon processes and returning a token for each back to the application. It is the shared SSL daemon process that owns the actual TLS session. When a z/TPF application issues an API using a shared SSL session token, the z/TPF shared SSL code will route the request to the shared SSL daemon that owns that session.

The API will then be processed by the selected shared SSL daemon thread and eventually return to the issuing application. Figure 5-20 depicts the shared SSL daemons processing APIs issued by z/TPF applications.

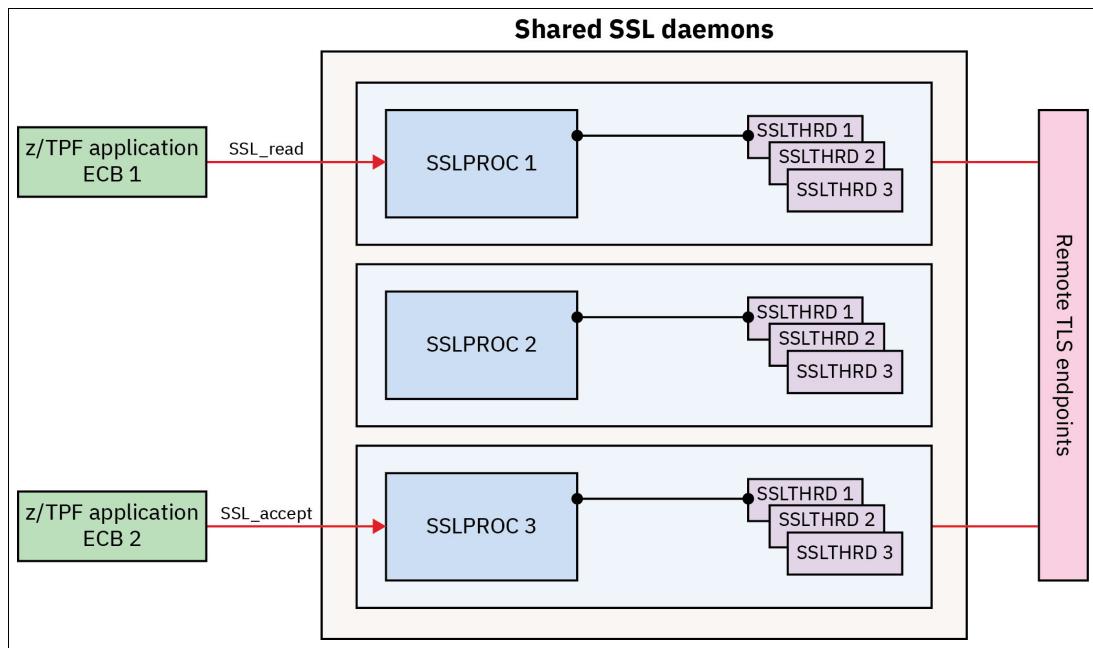


Figure 5-20 Shared SSL support

The steps shown in Figure 5-20 are as follows:

1. z/TPF applications in ECB 1 and ECB 2 issue the SSL_read and SSL_accept APIs.
2. The APIs are given to SSL daemons SSLPROC1 and SSLPROC3 respectively because the SSL daemon processes own the actual TLS sessions.
3. SSLPROC1 and SSLPROC3 will dispatch the work to their respective SSL daemon threads (SSLTHRDX) to process the APIs.
4. The APIs will interface with their remote TLS endpoints and return to the calling z/TPF application ECBs.

You can define many SSL daemon processes to be able to secure all your network traffic. The number of SSL daemon processes and threads can be configured as needed using the SSLPROC and SSLTHRDX parameters in Keypoint 2 (CTK2).

The shared SSL daemon configuration is very flexible. For each TLS server application on z/TPF, you can have all sessions with that application go to one specific shared SSL daemon or be load balanced across all the shared SSL daemons by updating the /etc/sslshared.txt file. One SSL daemon can manage sessions for multiple applications or can be dedicated to one application.

5.2.2 Securing data at rest

Securing data at rest involves encrypting any data that resides on a storage medium. For example, data that resides on disk (DASD) is the most common example of data that is at rest. Many regulations focus on protecting data at rest because attackers consider it a highly valuable target, especially if all the data in a database can be stolen. IBM Storage solutions provides the capability for disk level encryption. For more information, see [Disk encryption](#).

While disk level encryption will encrypt data written to disk, it may not be sufficient in many current and future regulations. For example, disk encryption alone does not protect the data flowing between the IBM Z processor and DASD control units or data cached in the z/TPF database cache in memory (VFA), which may exist in the memory cache for weeks or even months. In other words, data at rest includes more than just the data as it is written to disk.

When encrypting data at the z/TPF application layer or at the z/TPF database access layer, the data is encrypted before it is given to the DASD control unit. This approach ensures that:

- ▶ The data is encrypted while it resides in the z/TPF database cache, or Virtual File Access (VFA).
- ▶ The data is encrypted when it flows down the fiber connections to the DASD control units and the DASD itself.
- ▶ The data remains encrypted if it is replicated into a DR site, with the secure keystore being replicated as well containing the keys to unlock this data.

Figure 5-21 depicts data being encrypted and highlights all the areas (in green) where the data remains encrypted.

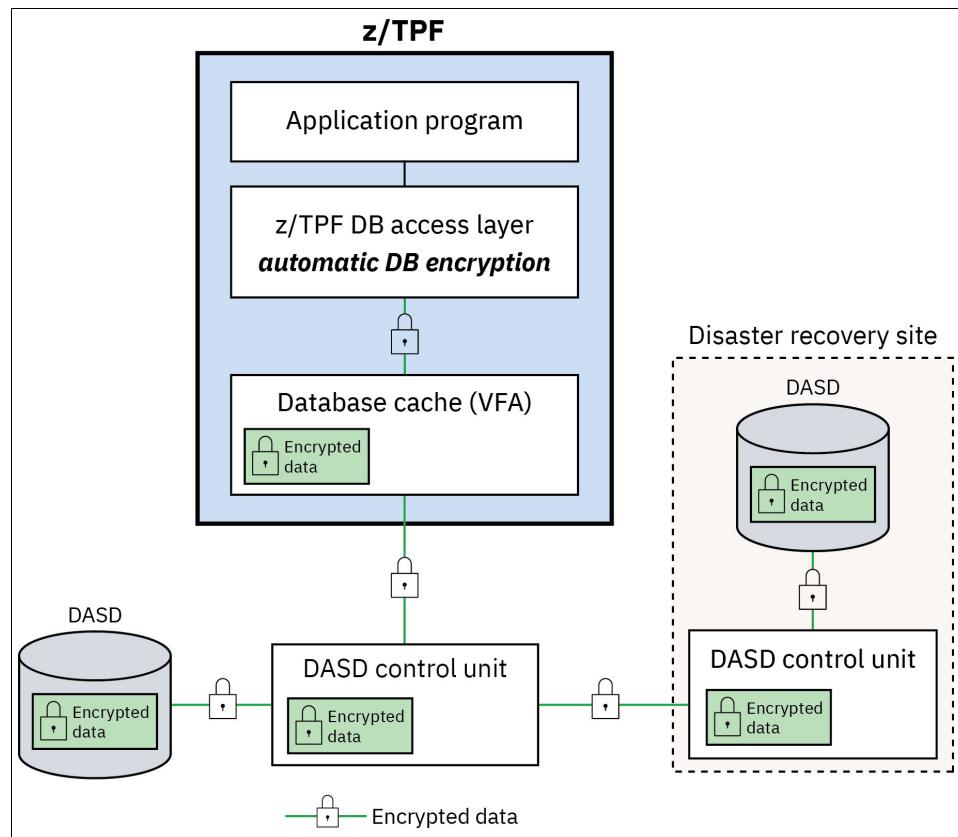


Figure 5-21 z/TPF database encryption

Data encryption within your application

A z/TPF application can encrypt data using quantum safe ciphers and keys from the z/TPF keystore by issuing the `tpf_encrypt_data` API. An application can subsequently decrypt that data by issuing the `tpf_decrypt_data` API. Symmetric key encryption is normally used to encrypt data that resides in the database. Symmetric key encryption is designed for bulk data encryption and decryption that needs to be done at scale for transactional purposes.

When you create a symmetric key in the z/TPF keystore, you assign two names to that key. The encryption key name is used to perform encryption using that key. The decryption key name is returned on the `tpf_encrypt_data` API and is used as the name to perform decryption of the data that you just encrypted. Saving the decryption key name along with the encrypted data allows you to rotate symmetric keys over time without affecting z/TPF applications using those keys. In other words, you can change/rotate keys without having to change your z/TPF application programs. It also allows you to not require any application downtime while you transition from one key to another.

The following example describes a scenario where an application is working with a reservation record and encrypting the payment information within that record. First, using the **ZKEYS CREATE** operator command you create a key in the keystore with an encryption key name of MYKEY1 and a corresponding decryption key name of MYDKEY1 as depicted in the following example. The key depicted in this example is the active key for this encryption key name, meaning this is the key that would be used for application encryption operations that specify that encryption key name (MYKEY1). Note that the secret key column is the actual key value that would never be displayable.

Encryption	Decryption			
Key Name	Key Name	Active	Cipher	Secret Key
MYKEY1	MYDKEY1	YES	AES256CBC	"KEY1"

Figure 5-22 shows the application using the `tpf_encrypt_data` API and the MYKEY1 encryption key name to encrypt the Payment Information in the reservation record.

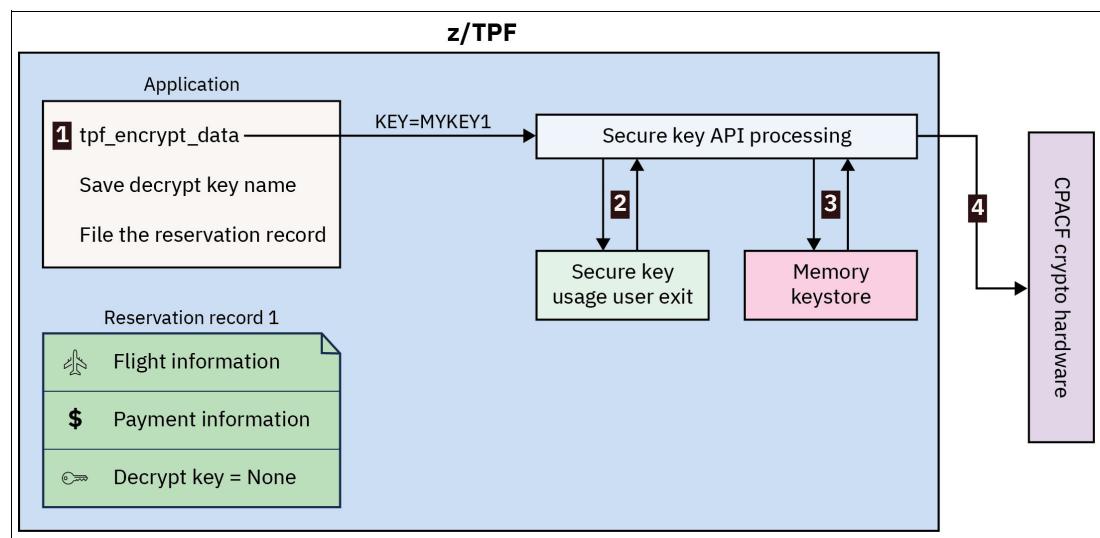


Figure 5-22 Secure keystore example 1

The steps in Figure 5-22 are as follows:

1. The z/TPF application issues the `tpf_encrypt_data` passing an encryption key name of MYKEY1 to encrypt the payment information in the reservation record.
2. The secure key usage user exit is called to validate that this z/TPF application program has access to use the symmetric key called MYKEY1.
3. The active entry for MYKEY1 is located in the z/TPF symmetric key memory keystore.
4. The IBM CPACF cryptographic hardware accelerator is used to perform the encryption operation on the payment information using the key value for the symmetric key with the name MYKEY1.

Figure 5-23 shows the steps upon return from the encryption operation in CPACF and subsequently return from the tpf_encrypt_data API.

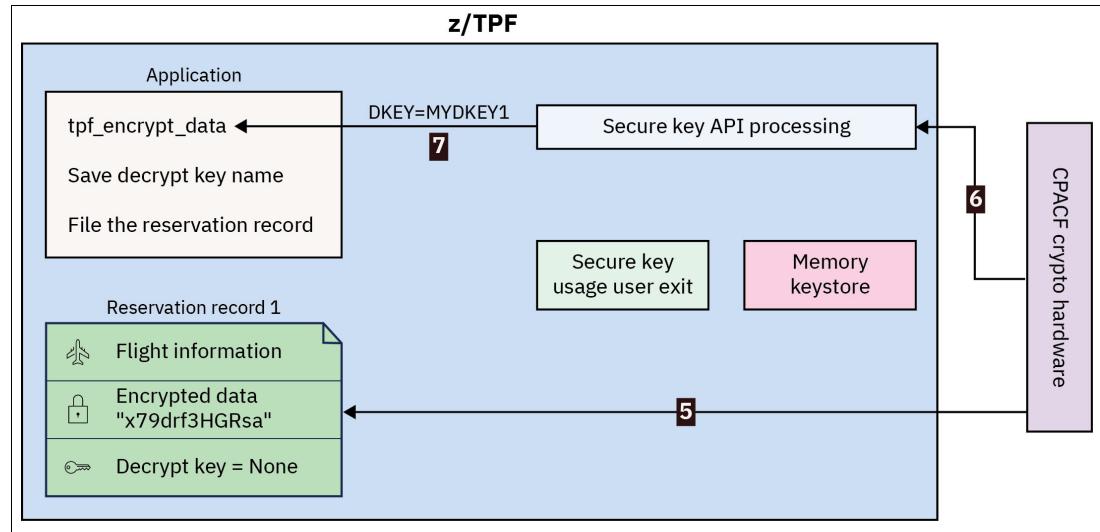


Figure 5-23 Secure keystore example 2

The steps in Figure 5-23 are as follows:

5. The encrypted data is saved in the location specified on the tpf_encrypt_data API, in this case the payment information within the reservation record.
6. Control is returned to the secure key API processing.
7. The decryption key name, MYDKEY1, which is what will be used to decrypt the encrypted payment information in this record is returned to the application.

Figure 5-24 shows the steps the application must take to ensure the encrypted data can be decrypted.

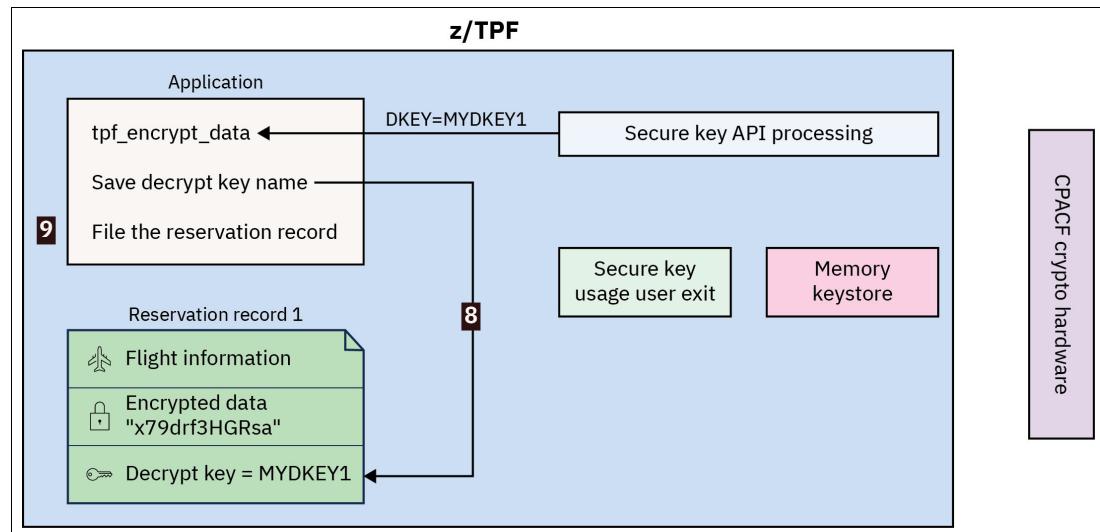


Figure 5-24 Secure keystore example 3

The steps in Figure 5-24 on page 40 show the following:

8. The decryption key name, MYDKEY1, returned on the tpf_encrypt_data API is saved in the reservation record along with the encrypted payment data information.
9. The reservation record containing the encrypted data and the decryption key name to decrypt that data is filed down to disk for use on subsequent transactions.

Saving the decryption key name allows administrators to rotate symmetric keys over time without having to update the applications using that key. By defining a new key with the same encryption key name, you can have some data encrypted with the key value associated with encryption key name MYKEY1 and decryption key name MYDKEY1 and some data encrypted with a new key (the value associated with encryption key name MYKEY1 and decryption key name MYDKEY2). In the following example, a second symmetric key has been created with the same encryption key name MYKEY1, but a different decryption key name MYDKEY2. This newly created key is now the active symmetric key for encryption key name MYKEY1; therefore, all subsequent encryption requests will use the newly created key. Note that the actual key value, which would never be displayed, is different.

Encryption Key Name	Decryption Key Name	Active	Cipher	Secret Key
MYKEY1	MYDKEY1	NO	AES256CBC	"KEY1"
MYKEY1	MYDKEY2	YES	AES256CBC	"KEY2"

Now when an application issues a tpf_encrypt_data API, passing key name MYKEY1, the data will be encrypted using a different key value and a different decryption key name is returned, MYDKEY2. Figure 5-25 shows the application encrypting the payment information of a different reservation record. Since the data is encrypted using a different key value, a different decryption key name is returned.

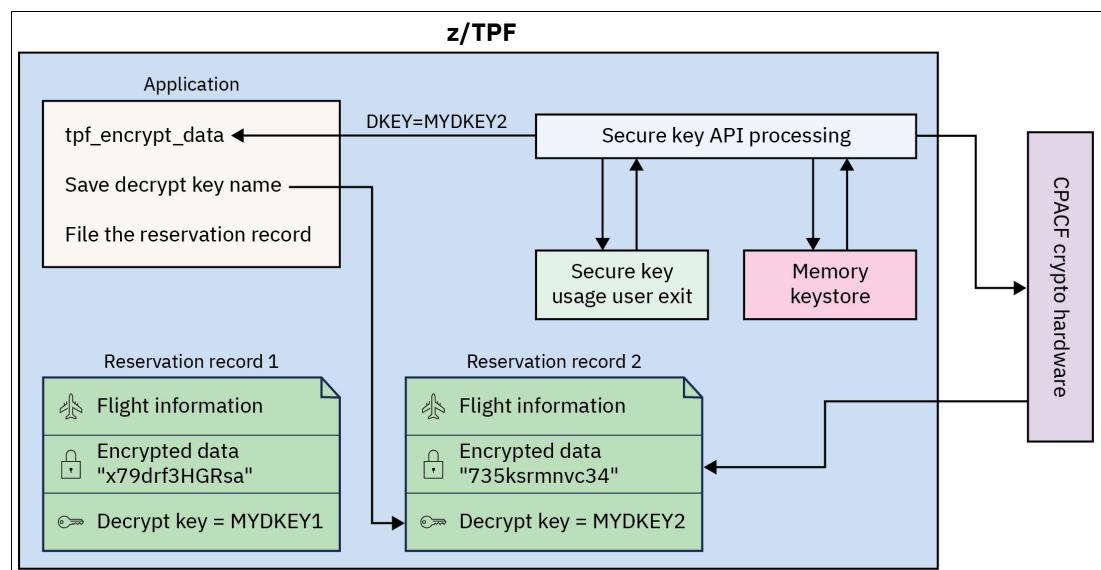


Figure 5-25 Secure keystore example 4

Now you have a database that has encrypted data with the key value for encryption key name MYKEY1 and decryption key name MYDKEY1. While you have other encrypted data with the key value for encryption key name MYKEY1 and decryption key name MYDKEY2.

The key with encryption key name MYKEY1 and decryption key name MYDKEY1 must remain in the keystore until there is no longer data encrypted using that key. This may occur naturally over time if the data is only present in your database for a fixed period, or application utilities can be run to re-encrypt all the data using encryption key name MYKEY1, which will force it to be re-encrypted using the key value associated with decryption key name MYDKEY2.

When there are pieces of data that need to be encrypted in a z/TPF database, a best practice is to encrypt all of the data in the database. There is enough cryptographic accelerator capacity in the IBM Z servers to encrypt all your data and achieve the following benefits:

- ▶ It is easier to prove to an auditor that all sensitive data is secure if all the data is encrypted.
- ▶ As regulations change, and additional data needs to be encrypted, there is nothing you need to do.
- ▶ Encrypting all the data is less complex and less error prone than cherry-picking pieces of data to encrypt.
- ▶ More overhead is consumed to encrypt several small pieces of data rather than one large block of data because there is a fixed startup cost required to interface with the cryptographic hardware.

Encrypting data in traditional z/TPF databases (find and file APIs) is the responsibility of the application program. However, if you are using z/TPFDF, you can use automatic z/TPFDF encryption support where the z/TPFDF database access layer code performs all the encryption and decryption. In other words, your applications work with data in the clear and are not involved with or even aware that encryption is being used.

Enabling a z/TPFDF database for encryption can be done while the application is up and running and processing transactions against that database (no database downtime).

z/TPFDF database encryption

When using z/TPFDF as your database manager, you can enable a specific z/TPFDF database for encryption without requiring any application changes or outages. A database can be enabled for encryption using quantum safe ciphers while the system is up and running and transactions are being performed on that database. z/TPFDF encryption uses keys securely stored in the z/TPF keystore to encrypt the data. The fact that data is encrypted is transparent to applications. For example, when an application program issues a z/TPFDF programming API to read data from the database, the data returned to the application is unencrypted. Similarly, when for diagnostic purposes a z/TPF operator needs to use the **ZUDFM** command to display the contexts of a specified z/TPFDF subfile, the decrypted data is displayed.

A typical z/TPFDF subfile, for example a reservation record, is a chain of 4K records on disk with a prime block as the head of chain, followed by zero or more overflow blocks. Each 4K record contains a z/TPFDF header and trailer, and the remaining area of the 4K record is reserved for user data. This data would consist of one or more z/TPFDF logical records. Figure 5-26 on page 43 shows a typical z/TPFDF user data format.

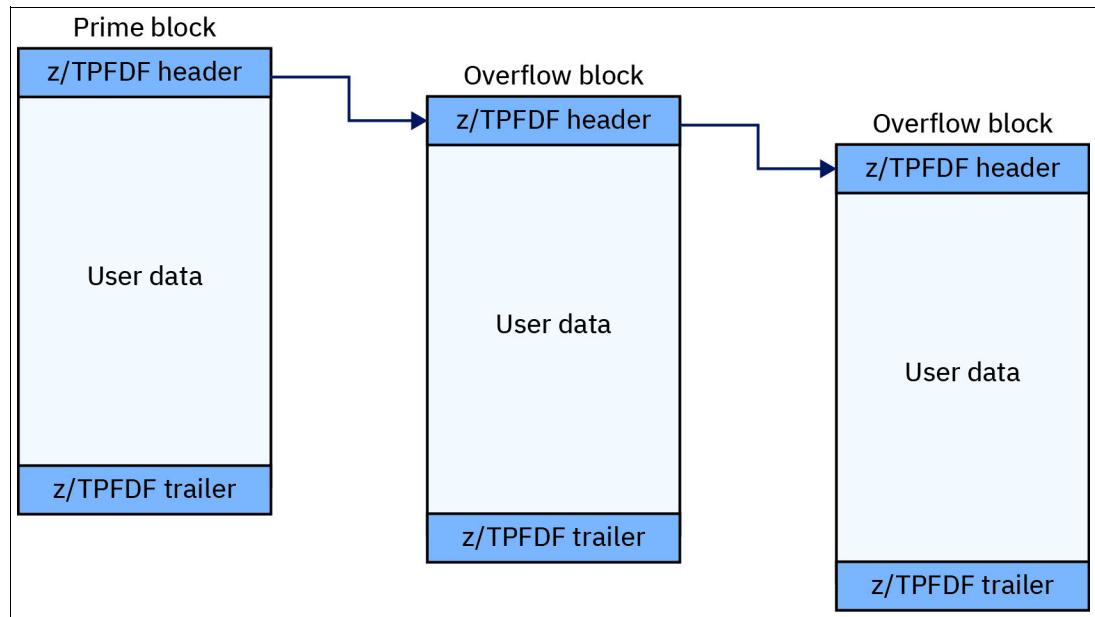


Figure 5-26 Typical z/TPFDF subfile format

Before you enable z/TPFDF encryption on a z/TPFDF database, you must migrate the z/TPFDF database to add the new z/TPFDF encryption trailer format. This is the area in which z/TPFDF would save the decryption key name when z/TPFDF encryption is enabled for this z/TPFDF database. You can migrate to the new z/TPFDF encryption trailer format using the CRUISE utility with the pack and migrate parameters specified. Figure 5-27 shows the same z/TPFDF user data format with the new trailer added.

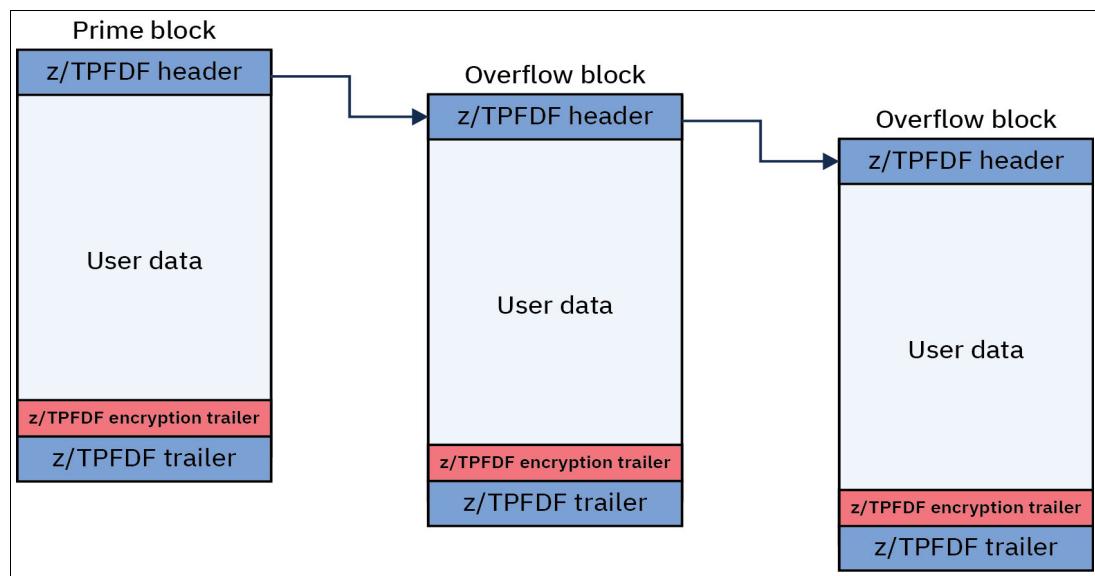


Figure 5-27 Typical z/TPFDF subfile format with an added trailer

While the chain of 4K blocks together make up a unit of database information, each 4K record is read from disk and filed to disk individually. In fact, in each transaction, all of the 4K records do not need to be read from disk, only those needed to complete the transaction are read from disk. The z/TPFDF encryption works at a 4K record level. Only those 4K records that are read from disk are decrypted and those that are filed down to disk are re-encrypted.

Since z/TPFDF takes care of saving the decryption key name along with each 4K record, a database administrator can seamlessly rotate keys, while the application is active processing transactions against the database. In fact, each 4K record that composes the subfile might be encrypted with different keys.

Figure 5-28 highlights the same z/TPFDF data with z/TPFDF encryption enabled.

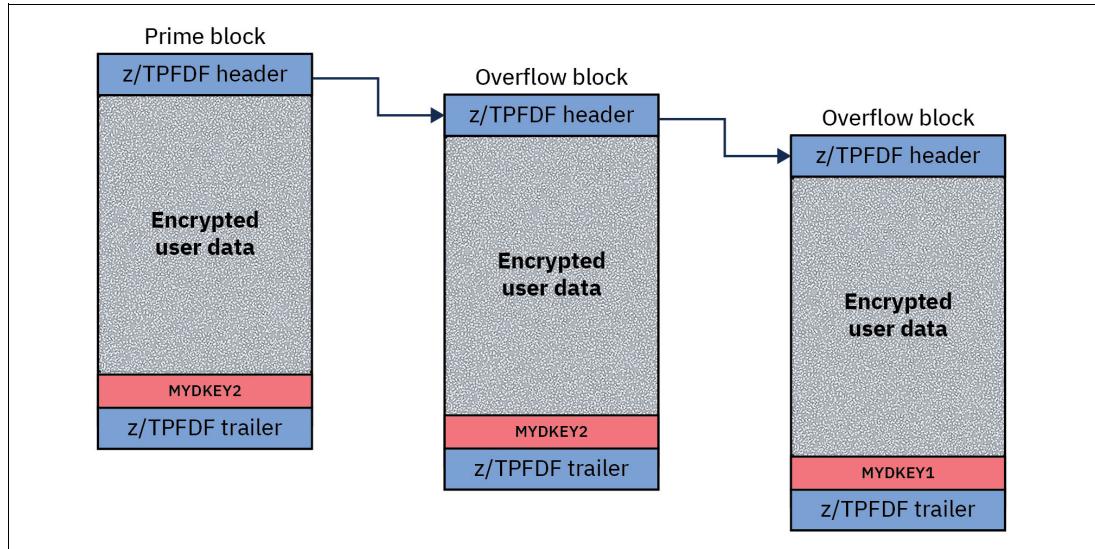


Figure 5-28 Typical z/TPFDF subfile format enabled for encryption

After enabling z/TPFDF encryption or you rotate a specific key, you can run the CRUISE utility with the PACK option against that z/TPFDF database to ensure all the data within that z/TPFDF database is encrypted or re-encrypted. When rotating keys, after running the CRUISE utility with the PACK option, you can be assured that the everything has been re-encrypted using the newly created key and the previous key used for encryption can be deleted.

In summary, you can enable z/TPFDF encryption by completing the following steps. These steps can all be completed with zero system or application down time.

1. Update your z/TPFDF database definitions (DBDEF) to enable the z/TPFDF file for encryption.
2. Using the CRUISE utility with the pack and migrate parameters specified, you can migrate the z/TPFDF file to create the new z/TPFDF encryption trailer format.
3. Using the ZKEYS command, define and activate a new quantum safe symmetric key.
4. Using the ZUDFM command, define and enable z/TPFDF encryption for the z/TPFDF database.
5. Using the CRUISE utility with the PACK parameter, encrypt all the data in the z/TPFDF database.

IBM tape encryption

It is common practice for z/TPF applications to log data to tape when that data must be retained for a period (30 days, 90 days, and so on) because of regulatory requirements or internal company policy. Typically, only a handful of that data logged to tape is ever looked at. For example, the data is looked at if there is a dispute over whether a transaction was processed or in some cases to debug an application problem. Tape is the most cost-effective method for logging and retaining large amounts of data where only a tiny fraction of the data is ever used after being written to tape.

When encrypting data at the z/TPF database layer that same data is read by the same z/TPF processor or another processor within a loosely coupled complex. Therefore, use of the z/TPF keystore makes sense because the process reading the data has access to the z/TPF keystore. However, how do you encrypt application log data being written to a tape drive? Generally, tape data written on z/TPF is read by an entirely different platform, like IBM z/OS or Linux on IBM Z. This data being written to tape might contain sensitive data. When writing sensitive data to tape, a best practice is to use tape encryption. For more information, see [Tape Encryption Overview](#).

5.2.3 Protecting data in use

We have described the capabilities of protecting your data in flight across the network and when the data is at rest on disk or tape. But how do you protect your data while it is in use during the processing of a transaction on z/TPF? For example, say an application reads sensitive z/TPFD data from disk into local heap storage within the application process or ECB. How can you protect that sensitive data while it resides in memory? What happens if that application process encounters an error that results in a system error dump? How do you prevent that sensitive data from being in the clear in z/TPF system dumps? How do you prevent sensitive data in memory from being displayed by an operator using z/TPF commands?

One way this can be accomplished is using the z/TPF nondisplayable storage capability. Your application issues an API to mark the storage containing sensitive data in the clear as nondisplayable. Marking storage as nondisplayable means that storage will not be visible in memory displays using z/TPF functional commands, in system errors or dumps, or from within the TPF debugger. In all these cases the contents of the memory that is marked as nondisplayable is shown as asterisks (*). Application programs mark an area of storage as nondisplayable using the `tpf_ndsp_mark` API.

Say you have read a reservation record into a core block within the application space. This reservation record contains some encrypted sensitive data. Before decrypting this data in place, you can mark the area containing the encrypted data as nondisplayable to protect the data while it is in use. Figure 5-29 on page 46 shows a reservation record containing encrypted sensitive data. The application would mark this area as nondisplayable using the `tpf_ndsp_mark` API passing a pointer to the area and the length of the area.

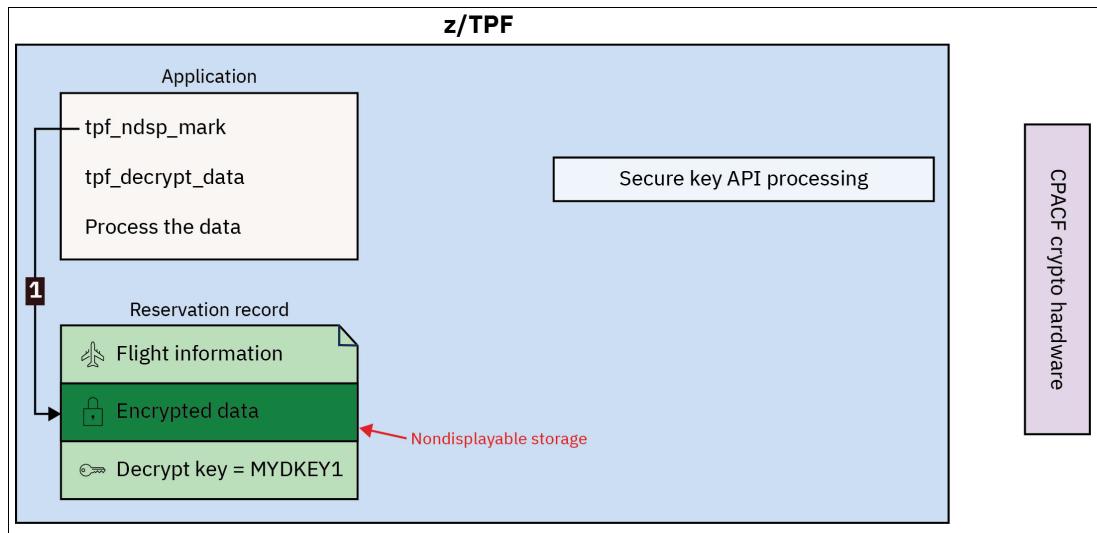


Figure 5-29 Reservation record with encrypted sensitive data in memory

Once the data is marked as nondisplayable, the application can now decrypt the data in place. In Figure 5-30, the application issues the `tpf_decrypt_data` API passing the decryption key name saved in the reservation record. The secure key API processing interfaces with the IBM CPACF hardware to perform the actual decryption operation. In this example, the sensitive data contains the passenger's address, but could contain several different things.

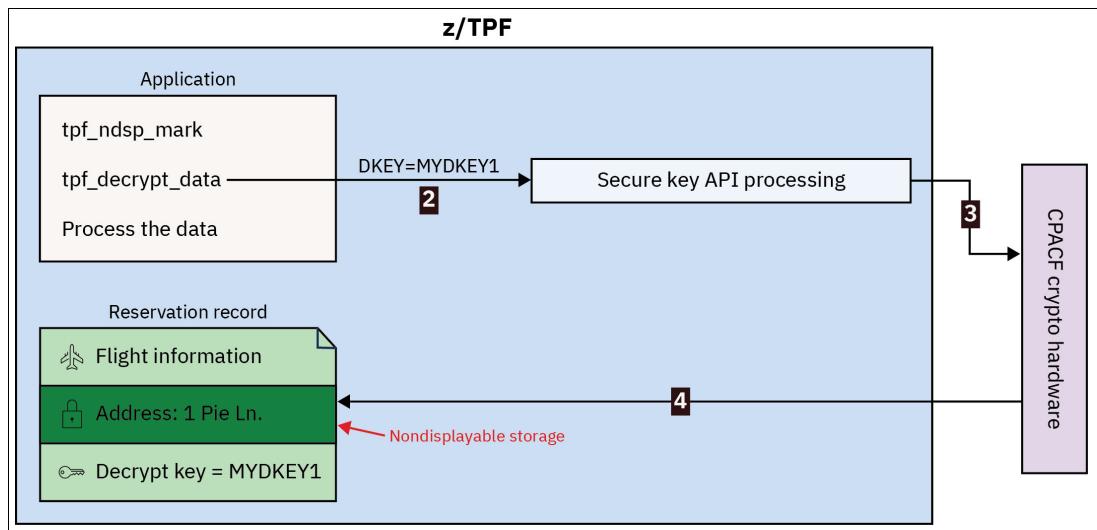


Figure 5-30 Application decrypting data in memory in nondisplayable storage

After the decryption of the sensitive data occurs, the data is in the clear for the application, but is not displayable by any other methods. In Figure 5-31 on page 47, the application changes the passenger's address within the reservation record.

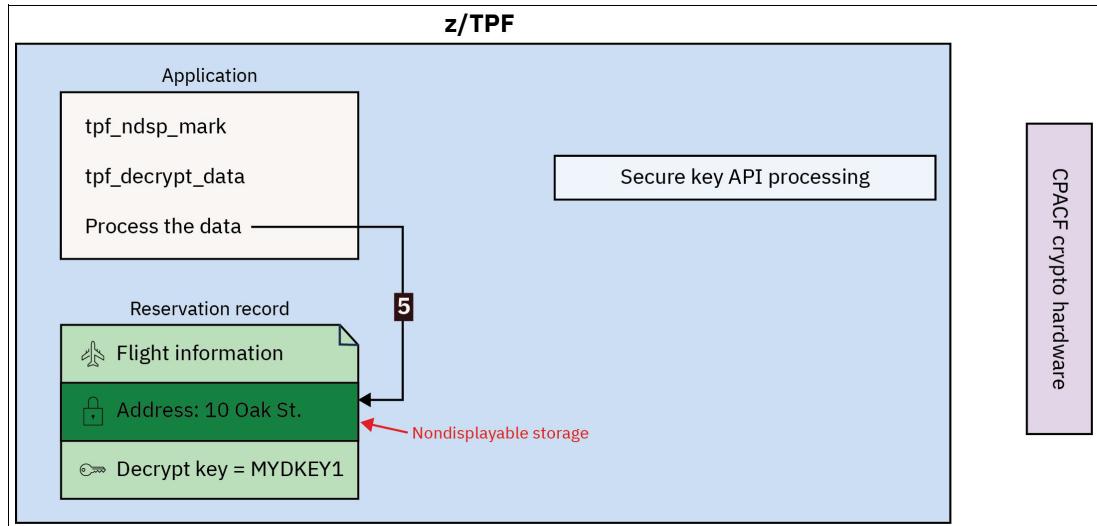


Figure 5-31 Application changing an address within a reservation record in memory

At this point, the application has completed processing and making changes to the reservation record. The application now issues the `tpf_encrypt_data` API to re-encrypt the sensitive data within the reservation record. Figure 5-32 shows the application encrypting the sensitive data. While it is not shown, the decryption key name would be returned on the `tpf_encrypt_data` API and saved in the reservation record.

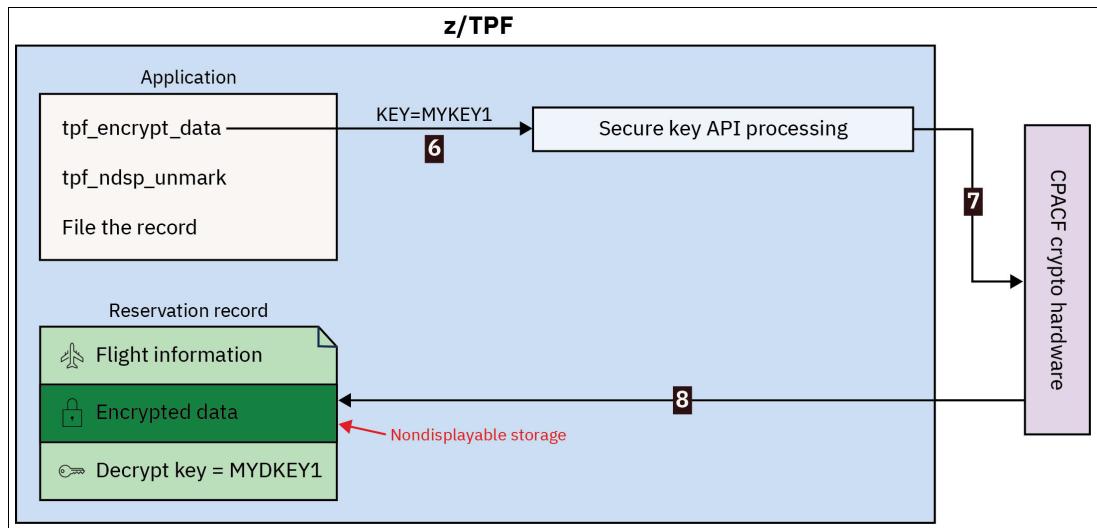


Figure 5-32 Application encrypting sensitive data in memory

At this point, any sensitive data has been re-encrypted. Figure 5-33 on page 48 shows the application issuing the `tpf_ndsp_unmark` API allowing the storage containing the encrypted data to be displayable again. Now the application can file down the changes made to the reservation record to complete the transaction.

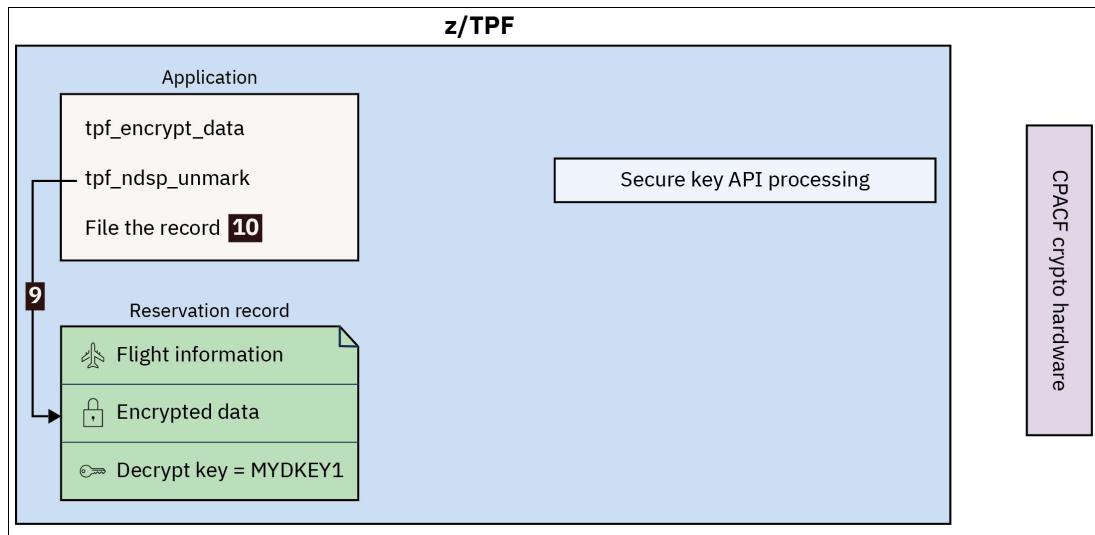


Figure 5-33 Application allowing storage with encrypted data to be displayable

The z/TPF nondisplayable storage feature allows you to protect your sensitive data while it is being processed by an application. With nondisplayable storage, you can be assured that your sensitive data is only viewable by the z/TPF application processing the data.

Another aspect that protects the data that is in use on z/TPF is true transaction isolation. On other platforms, as shown in Figure A-1 on page 90, there are separate, long running, and multi-threaded processes, and each one handles different functions. With separate long running processes handling each function, like HTTP layer process or database server process, each transaction flows through each long running process. In this model, a given transaction is sharing the same memory space as the other transactions flowing through the process. In the multi-threaded, long running process model on other platforms, all threads within a process share the memory, so one transaction can see the data of other transactions that are handled by other threads in the same process.

As shown in Figure A-2 on page 91, each typical transaction on z/TPF runs in its own single-threaded process. In the z/TPF model, true transaction isolation is achieved because each transaction runs in its own process. Since each transaction is isolated in its own process, it can see only its own data, not the data of any other transaction.

5.2.4 Protecting your data from corruption

While securing your data in flight, in use, and at rest protects your data from exposure, it does not necessarily protect data from corruption. Data corruption refers to any event that causes data loss or changes data in some way as to make it incorrect or unusable. For example, a malicious actor might not be able to read the data because it is encrypted, but they might corrupt the data by writing random data over the encrypted data. In another example, an employee might accidentally delete data while making necessary updates to your database.

As part of your DR and high availability (HA) solutions, it is a best practice to continuously replicate z/TPF data to other sites. For clients using a geographically dispersed active-active architecture, the data is replicated to other active z/TPF systems within seconds to keep the databases across the systems as up to date as possible. For DR sites, DASD copy service functions like IBM Global Mirror also copy the data from the primary site to the disaster recovery site within seconds to minimize data loss in the case of a disaster.

While HA and DR solutions are designed to replicate data within seconds, it also means corrupted data is quickly propagated to other active z/TPF systems and the disaster recovery sites. As a result, continuous replication of your data cannot be used to recover from corruption and another method is needed to secure your data in case of corruption.

To preserve data in case of corruption, you should maintain multiple point-in-time (PiT) copies of your data. Each PiT copy or backup is a snapshot of your data at a moment in time, allowing you to recover your data to that moment. In case the recent PiT copies are corrupted, multiple copies should be maintained so you can recover from older copies that might not be corrupted.

Since the advent of DASD copy services, many clients used DASD copy services functions like IBM FlashCopy® to create point-in-time copies. For example, in Figure 5-34, a z/TPF client might design their solution to maintain up to 3 PiT copies of their data stored on sets of backup volumes F1.a, F1.b, and F1.c. Each set of backup volumes F1.x contains data from a set of source volumes H1 for the same point in time. In this example, backup set F1.a could be from last night (time t3), F1.b from two nights ago (time t2), and F1.c from three nights ago (time t1). Because they only have room for 3 copies, a client would overwrite the oldest backup when creating a new backup. When they take the next backup tonight at time t4, the new copy will overwrite the copy t1 on backup volumes F1.c, leaving the client with copies from times t4, t3, and t2. Similarly, the next copy taken at time t5 would overwrite the copy t2 on backup volumes F1.b. The frequency of making copies and the number of copies maintained is a business decision, weighing the cost of the copies with the risk of having insufficient or too infrequent copies.

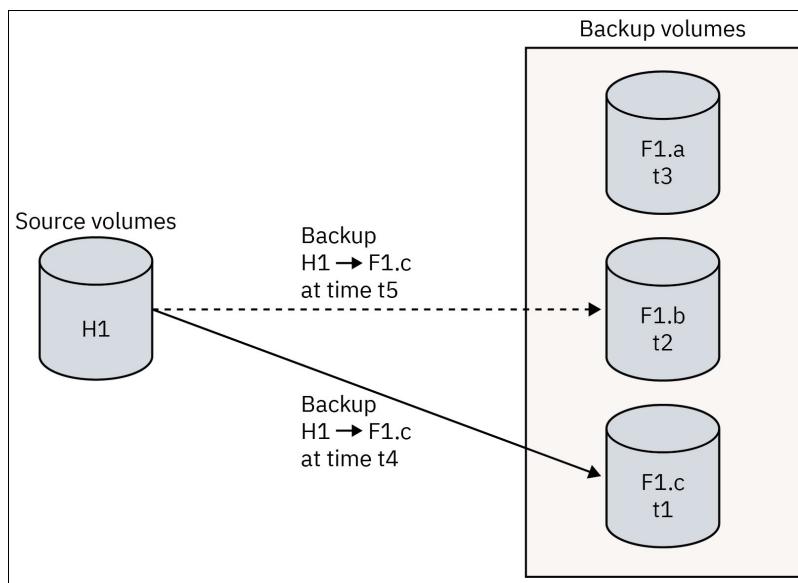


Figure 5-34 Creating point-in-time backups to separate sets of backup volumes

While these PiT copies could be used in case of corruption of your production data, the copies themselves need to be secured. These types of copies are stored on regular volumes in the DASD control units and the backup volumes could be accessed, modified, or deleted in various ways. Other systems could be used to mount the backup volumes, access the PiT copy of the data, and either accidentally or intentionally corrupt the data. In another example, an employee making a needed configuration change to the DASD control unit might accidentally alter the definition of the volumes, causing some of the PiT copy of the data to be lost.

To secure your data in case of corruption and protect backups from unintended or unauthorized manipulation, clients should use cyber resilient solutions to maintain their PiT backups. For clients using the IBM DS8000 family of storage systems, this type of solution can be provided by the IBM Safeguarded Copy function.

As with IBM FlashCopy, IBM Safeguarded Copy creates multiple PiT backups, allowing you to restore to the most recent, uncorrupted copy of your data. However, IBM Safeguarded Copy also secures your backups in several different ways. Instead of storing the data on regular volumes, IBM Safeguarded Copy stores the backup data in a storage space called IBM Safeguarded Copy Backup Capacity (SGBC). IBM SGBC is separate from regular volumes and is not assigned device numbers or host addresses. As a result, IBM SGBC and the backups it contains cannot be accessed, changed, or deleted by host systems like z/TPF, z/OS, or other systems.

The SGBC can contain multiple backups, where each backup represents a separate PiT recovery point and can include multiple source volumes. This means IBM Safeguarded Copy can backup multiple source volumes at the same point in time and recover all those volumes back to the same recovery point. The number of backups depends on several variables, including the size of the IBM SGBC, and older backups can automatically expire to make room for the next backup. As a result, it is important to make sure the IBM SGBC has enough room to store all the cyber resilient backups required by your business.

Figure 5-35 shows a basic IBM Safeguarded Copy relationship between source volumes and the IBM SGBC containing your secure backups. In this example, the IBM SGBC contains four backups, with the backup at time t1 being the oldest and the backup at time t4 being the most recent. The next backup at time t5 would add a new backup to the IBM SGBC and, at that point, the IBM SGBC would contain five backups from times t1 to t5.

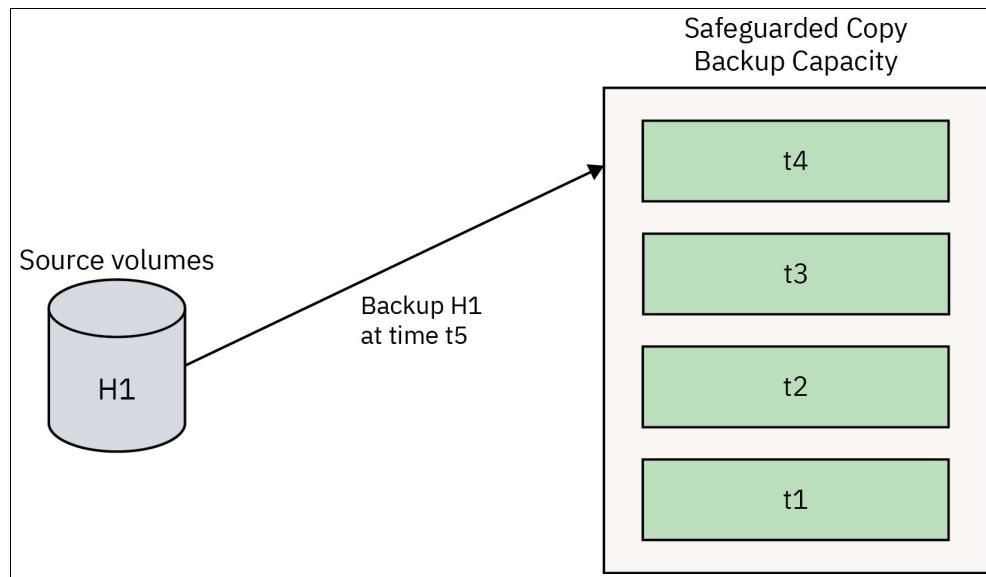


Figure 5-35 Creating secure PiT backups using IBM Safeguarded Copy

In addition to making IBM SGBC inaccessible to host systems, additional protections secure your IBM Safeguarded Copy backups from accidental or malicious manipulation. It is best practice to maintain separate roles for the DS8000 administrator and those responsible for IBM Safeguarded Copy. As a result, the DS8000 management interfaces DS CLI and DS GUI cannot be used to manage IBM Safeguarded Copy. A separate interface, such as IBM Copy Services Manager (CSM), must be used to manage IBM Safeguarded Copy.

IBM CSM supports multiple users with different roles and security levels, allowing you to isolate users and allow them access only to those IBM Safeguarded Copy sessions and functions they require for their job.

For added security, IBM CSM also supports dual control, which requires two users to initiate a task or action. This feature prevents one IBM CSM user from either accidentally or maliciously issuing destructive actions, like deleting IBM Safeguarded Copy backups or terminating IBM Safeguarded Copy sessions. Dual control should be used for copy services utilized in backup, disaster recovery, or high availability solutions critical to your business.

For more information, see IBM Redbook *IBM Storage DS8000 Safeguarded Copy: Updated for DS8000 Release 9.3.2*, REDP-5506-04.

5.3 Access control

5.3.1 Leveraging firewalls

Firewalls have been the first line of defense for controlling the access to your enterprise and systems within it for decades. A firewall is essentially a wall between your private networks and external public networks. The original intent of a firewall was to filter packets based on who is accessing (the remote IP address) and what they are trying to access (the port and local IP address). While firewalls have evolved over time to do more complex, deep packet inspection, the original intent of filtering packets remains in place today.

While most people view a firewall as an external device protecting your private network from external users, a firewall can also be software running on an individual machine inside your data center. The strongest protection is a combination of both internal and external firewalls within your enterprise, with external firewalls protecting you at the perimeters of your enterprise and internal firewalls protecting individual systems, like z/TPF.

Like most other platforms, the z/TPF operating system has a software-based firewall that allows you to prevent access to the z/TPF system using a set of coded rules. For example, you may have rules to only allow access to individual ports from specific remote users or networks. z/TPF IP packet filtering support provides the capabilities to control access to ports and protocols from remote networks accessing the system.

Say you have your REST services accessed through the z/TPF HTTP server on port 443. You only want access to these REST services to be provided through your API management layer and assume your API management layer is a set of machines on a specific network 198.51.100.0 / 24. You can setup rules to provide access to port 443 on z/TPF for network 198.51.100.0 / 24 while rejecting all others. For example:

```
ACTION-ALLOW PORT-443 FROM-198.51.100.0/24  
ACTION-REJECT PORT-443
```

Because the rules are processed sequentially until a match is found, we would allow access when the IP packets destined for port 443 are from remote IP address 198.51.100.0 / 24, while rejecting any other IP packets trying to access port 443. This is just one example of how you can use the z/TPF IP packet filtering support and the capabilities of it.

5.3.2 Designing and managing access to services

Services are a core part of maintaining your business needs through interacting with your z/TPF system. As such, your services must be designed and managed with security in mind.

It is a best practice to create a centralized API management (APIM) layer for your services, including services that reside on z/TPF, to securely control and manage access to all services in your enterprise. If you have a dozen different platforms that provide services, trying to manage access to services on each of those 12 platforms is tedious and error prone. For example, if an employee leaves the company or you terminate the contract with a business partner, you want to go to one place, a centralized APIM, to revoke access rather than trying to figure out which of the 12 platforms that user had access to and update each one. A centralized APIM gateway should be used to authenticate who the user is, and that the user is authorized to call the service in question. Password management is another reason why using a centralized APIM makes sense, rather than trying to keep user passwords in sync across all your platforms that provide services.

When creating z/TPF services, or services in general, it is important to implement fine grain access control and separate read only type services from those that can make updates. For example, instead of a single service that provides options to read or update a reservation record, you should implement two services: one to read a reservation record and the other to update a reservation record. Typically, many users are allowed to read reservation records, but only a limited number of users are allowed to update a reservation record. If read and update are combined into one service, then all users of that service are authorized to read and update, even if a given user should only be able to do read operations. By implementing two separate services, you can easily control and limit who can update reservation records.

While many users have a business need to read a reservation record, thus a business need to call a read reservation record type service, which parts of the data in that record should be returned to the user depends on the role of the user. For example, if the reservation record contains payment information like credit card information, most users do not have a business need to see that data. There are different ways to handle this. One way is to create different services based on role, such as a generic "read reservation record" service that most users are authorized to use and then a "read reservation record with payment info" service that only a few users are authorized to use. Another way to handle this is to have one read reservation record type service and pass the role of the user to the platform providing the service, like z/TPF, and then the service itself creates different output data (filtered) based on the role of the user.

General use services should have the ability to access one specific record or piece of information at a time, like one specific reservation record. You likely will also need to create services that return a fair amount of sensitive information, like an airline flight manifest or the list of passengers that flew into JFK airport yesterday, but only the subset of users with a business need for that data should be authorized to call services like that.

External services should be designed to require multiple inputs or pieces of information to access data. For example, an external service may return the passenger flight information for a passenger given a passenger's name, passport number, and flight reservation number. Services that access large amounts of information, like all passengers on a given flight, should be internal services only and should have restricted access on a need-to-know basis.

Services that update many database records that are only used in time of crisis might have access to significant amount of data, but these services should only be active or deployed when necessary, and inaccessible otherwise to all users.

5.3.3 Controlling access to data for z/TPF Java applications

As your enterprise deploys Java applications on the z/TPF system, concerns might be raised about the security of using Java applications and libraries from outside sources on z/TPF. z/TPF support for Java does not allow Java applications free reign to read and update traditional z/TPF or z/TPFDF databases. Note that Java applications do have access to the z/TPF file system, so it remains important to maintain access controls on the z/TPF file system.

Java applications running on z/TPF can only access traditional z/TPF databases (find and file APIs) and z/TPFDF databases by calling services that you create that allow Java code on z/TPF to call traditional z/TPF applications code (written in C/C++ and assembly) that access those databases. While z/TPF services support is designed to easily enable remote access to services on z/TPF, you can define a service on the z/TPF system that can only be called internally from a Java application that runs on the same z/TPF processor as the service.

By defining local stateful services, you can enable Java applications to access z/TPF and z/TPFDF databases but avoid exposing those services to remote systems. As a result, those services are guaranteed to only be called by code that is loaded to the z/TPF system. z/TPF includes an optimized interface between local Java code calling local services. Even though the Java code issues a REST API to call the local service, z/TPF bypasses the TLS and TCP/IP layers to avoid the network and encryption overhead associated with a remote service call. As a result, z/TPF support for Java offers both a performant and secure means for application modernization on the z/TPF platform.

5.3.4 Managing access for z/TPF support for MongoDB

z/TPF support for MongoDB allows remote access to z/TPFDF data by using a standard interface. By using z/TPF support for MongoDB, remote applications can use a standard, open-source [MongoDB client library](#) to directly read and update data that resides in a z/TPFDF database. Because z/TPF support for MongoDB provides direct access to potentially critical or sensitive z/TPFDF data (the same z/TPFDF data used by local z/TPF applications), care must be taken to control access to that data.

You must take explicit action to make a z/TPFDF database accessible through z/TPF support for MongoDB by creating and loading configuration files to the z/TPF system that define which z/TPFDF databases are accessible to remote users. By default, z/TPFDF databases are not accessible remotely by z/TPF support for MongoDB. You define which databases that remote MongoDB clients can access based on business need for that data. Figure 5-36 on page 54 shows how you can configure z/TPF support for MongoDB to expose only certain databases to remote clients. In this example, the PNR, Flights, and Inventory z/TPFDF databases can be accessed remotely, but the Payments z/TPFDF database cannot be accessed remotely.

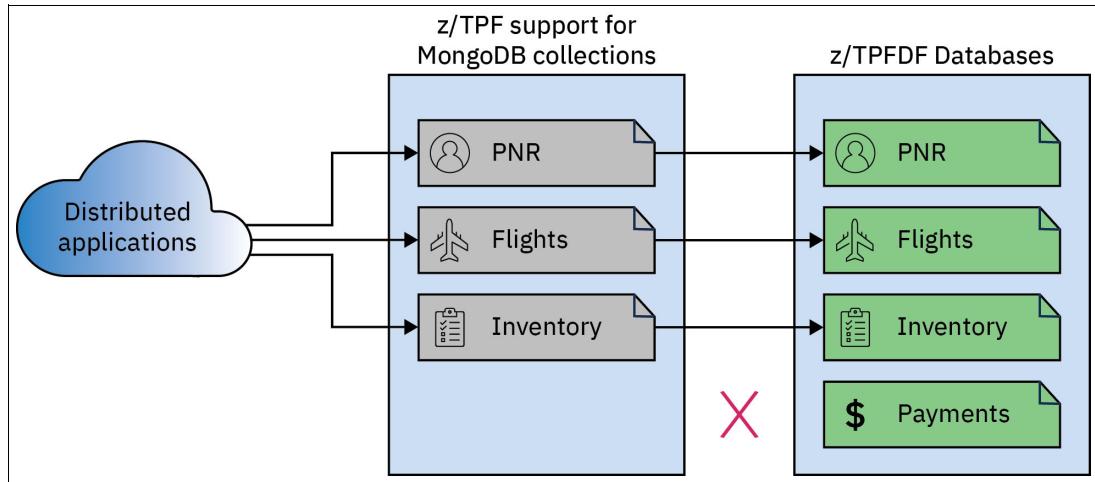


Figure 5-36 Configuration that exposes only certain databases to remote clients

After defining which databases are accessible by using z/TPF support for MongoDB, you can control access to each database by using the z/TPF user security database. The z/TPF user security database provides the capability for you to define users, roles, and privileges for MongoDB clients. With z/TPF support for MongoDB, each user can have one or more roles, and each role can have one or more privileges on specific databases defined.

For example, an administrator user might be configured with read and write access to all databases that are allowed to be accessed remotely, while other users might have read-only access to only specific databases. Figure 5-37 shows an example of how you can use users, roles, and privileges to limit access to your z/TPFDF databases based on the business needs of a set of remote users.

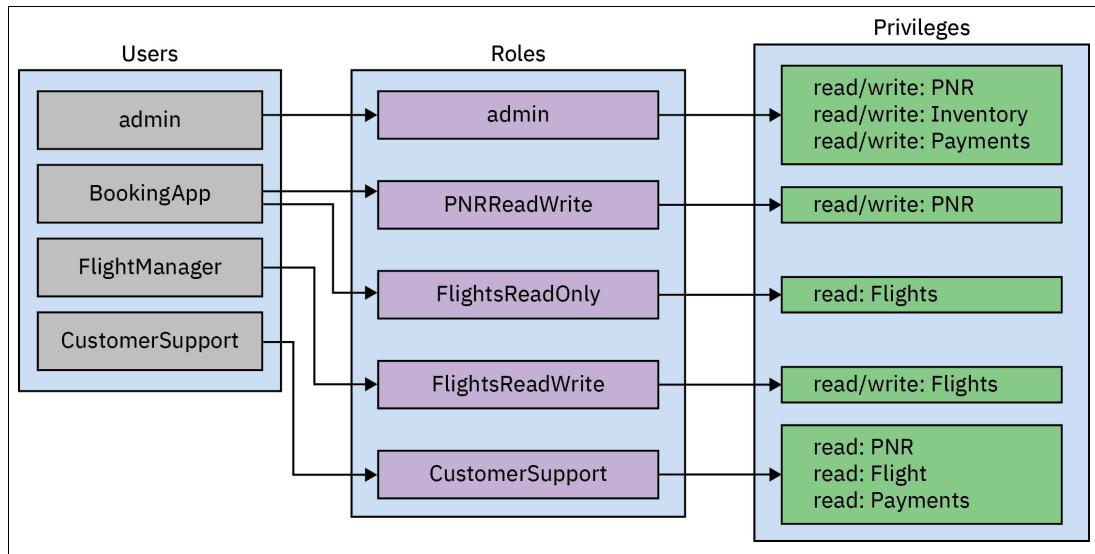


Figure 5-37 Using roles to control access to z/TPF databases for remote MongoDB users

You can use z/TPF operator commands to dynamically manage users and roles in the user security database. For example, if an error in a remote client application is causing errors on z/TPF processing requests from that user, the z/TPF operator can revoke that application's (user's) access to one or all databases in real time.

After databases and users are set up, a distributed application can use a MongoDB client library to access data through z/TPF support for MongoDB. To ensure that only trusted clients can connect to z/TPF support for MongoDB, you should use TLS client authentication with z/TPF support for MongoDB. In addition, a pre-request user exit is called before each MongoDB request, determining whether to accept or reject the request. For example, if the z/TPF system is running low on resources and this request is a low priority database query message, you can choose to reject the request.

z/TPF support for MongoDB requires client user authentication, meaning a remote client must provide credentials when logging in before that user is allowed to issue any requests to access z/TPF data. The z/TPF user security database implements standard password authentication methods supported by MongoDB clients.

5.3.5 Controlling access to data with z/TPF support for MongoDB

Multiple remote users might have permission to read a given z/TPFDF database, but only a subset of those users might have a business need to see sensitive information in that database. You can configure access control for specific data within records in a database. z/TPF support for MongoDB includes support for filtered collections. You can define a filtered collection with rules that control the visibility of certain data or logical records (LRECs) in z/TPFDF subfiles. For example, a z/TPFDF subfile might include three types of LRECs: passenger name records, passenger number records, and credit card records. If remote users do not have a business need to access sensitive credit card information, you can configure a filtered collection for z/TPF support for MongoDB so that the MongoDB clients can access the subfile, but the credit card records are filtered out of the response document sent by z/TPF. By using a filtered collection, you can limit the amount of sensitive information available to distributed users based on a need-to-know basis.

In addition to filtering certain LREC types out of a z/TPFDF subfile, you can filter records out based on the values of fields within that record. For example, a z/TPFDF subfile might contain data for several different airlines. Each LREC in the subfile might contain a field that associates the LREC with a particular airline. You can configure a filtered collection with filtering rules based on that identifier field, so z/TPF support for MongoDB only returns LRECs that correspond to a particular airline.

With z/TPF support for MongoDB, you can define multiple filtered collections that map to the same underlying database. Figure 5-38 on page 56 shows how you can configure multiple filtered collections, each used by a different user, where each collection filters out all data that is not required by the user. Access to each filtered collection is managed through the same system of users, roles, and privileges, enabling multiple users to use the same filtered collection without any additional administration overhead.

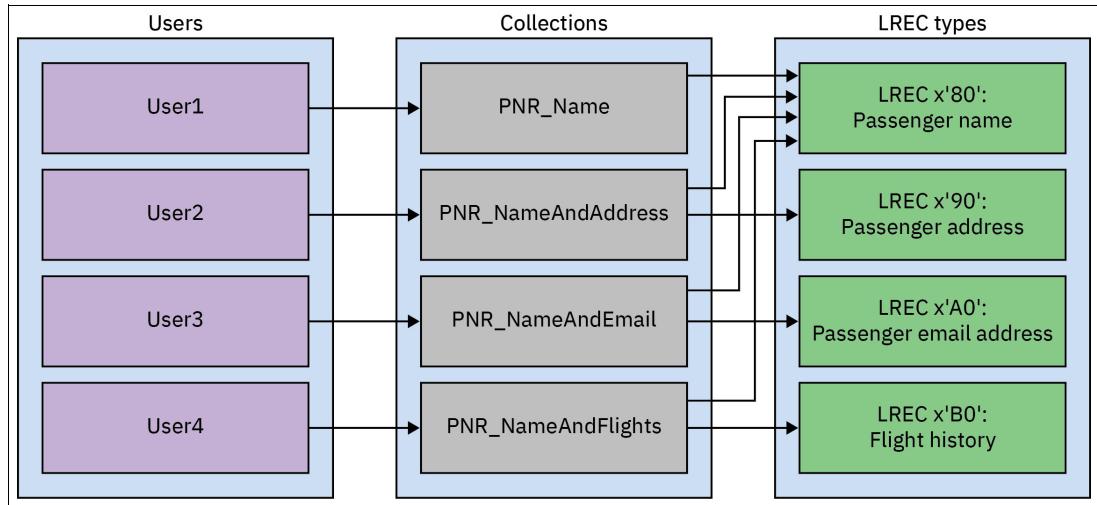


Figure 5-38 Using filtered collections to control which data remote MongoDB users can see

5.3.6 Controlling access to data with z/TPF data events

In many data breach situations, the data residing on the system of record is not the data that is breached. However, the replication of the data from the system of record to other business units and systems that need that data is what ultimately causes the breach. Controlling what data leaves the platform with the "need to know" mentality becomes important. What data does a given business unit need?

The z/TPF data events solution allows the application to focus on the business logic to process transactions rather than maintaining complex access control schemes. By using a database definition, you control:

- ▶ Which databases, when updated, trigger the system to generate a data event.
- ▶ Which remote systems get a copy of the data.
- ▶ How to filter and transform the data based on destination.
- ▶ Which transport mechanism to use to send the data event to that destination.

No z/TPF application changes are required to enable data events support or to change the data events configuration.

Figure 5-39 on page 57 shows an example where the database record that was created or updated is a hotel reservation record that includes payment information like credit card number and expiration date, and PII data like the guest's name, home address, and email address. A copy of the entire reservation record (PNR) is sent to the billing system because it has a business need for all the data. The payment information is filtered out and not sent to the customer support system because they have no business need for that data. All payment information and PII data is filtered from the copy of the data that is sent to the analytics system because that system has no business need for any of that data.

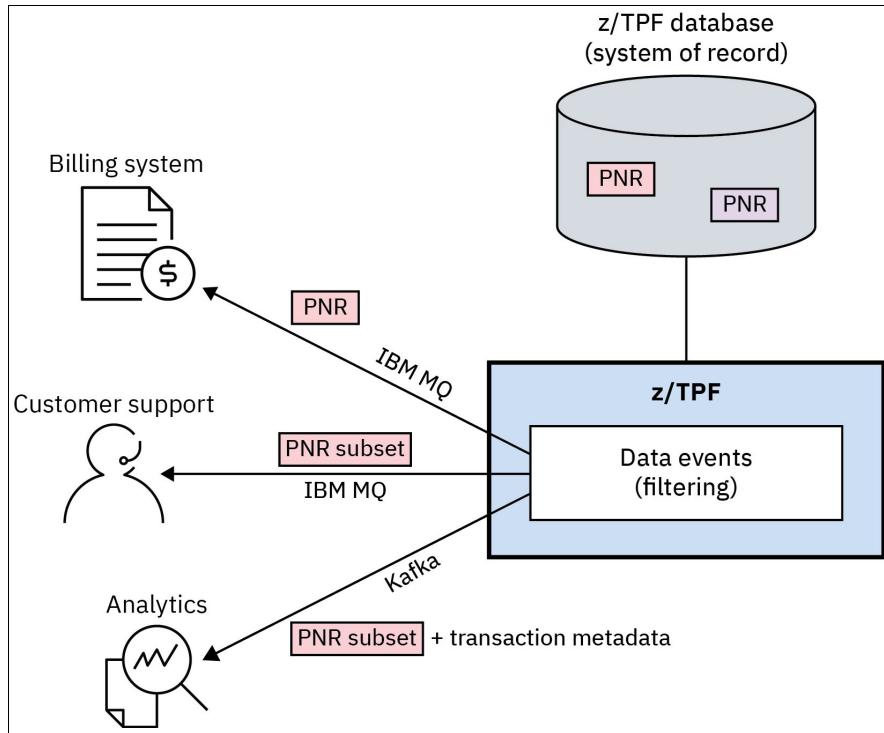


Figure 5-39 Filtering and augmenting data sent to remote data consumers

Using this approach, you can control which data is replicated to other platforms and for what purpose, limiting the exposure of sensitive data to only business units that require that data. Sending data off the z/TPF system in this manner is more secure because you are limiting who has a copy of sensitive data in your enterprise.

5.3.7 Managing access to the z/TPF file system

The z/TPF file system is a file system that is compatible with the Portable Operating System Interface (POSIX), including the same permissions schema that exists on a Unix or Linux system. A key aspect of a POSIX file system's access control mechanism is the use of users and groups to manage access to files and directories within the file system. The z/TPF file system supports the same authorization mechanisms to restrict access to data to only the users that need it. As with a Unix system, you can set owners, groups, and permissions such that sensitive files or directories are only accessible by root or administrator users.

z/TPF file system security support takes advantage of file permissions to provide an easy and effective access control mechanism for file system files. You can use file system security support to ensure that operators other than the primary console, or prime computer room agent set (CRAS), do not have root access to the z/TPF file system, and can only access the files that they need. Because the prime CRAS terminal is your primary administrator console, the prime CRAS always runs as root, even with file system security support enabled. Because prime CRAS runs as root, you still have the capability to perform all administrative tasks using restricted and sensitive files.

Note that on most other platforms, the file system contains user code (applications), databases, and system code. In other words, everything is in the file system. That is not the case on z/TPF - traditional z/TPF databases (find and file APIs) and z/TPFD databases are not in the file system. Systems code and application code is not in the file system either, with the exception being Java code.

The z/TPF Java environment can be configured to only allow execution of JAR files from a part of the file system that only the z/TPF program loader can update. The primary use of the file system on z/TPF is for configuration files.

5.3.8 Managing access for z/TPF operators

The z/TPF operator console is a powerful tool for system administration, monitoring, and maintenance. Because the z/TPF operator console provides you with the ability to display memory and databases, and in some cases update memory and databases and load code, it is important to maintain proper access controls for z/TPF operators. By using TPF Operations Server and file system security support on operator commands, you can control which commands a given operator or coverage programmer is allowed to use. It is best practice to limit access to commands that can alter the z/TPF system. However, you also need to make sure certain people have access to be able to debug and take corrective actions in a crisis.

The IBM TPF Operations Server product provides you with a way to manage access to the z/TPF system from multiple terminals (users). Access control with TPF Operations Server begins with the client authentication user exit. You can code the TPF Operations Server client authentication user exit to authenticate users each time they connect. Because the client authentication user exit is implemented by your TPF Operations Server administrator, you can use any authentication mechanism that makes sense in your enterprise.

For example, you might configure the client authentication user exit to authenticate a user ID and password against your enterprise Lightweight Directory Access Protocol (LDAP) server. The flexibility associated with TPF Operation Server authentication mechanisms allows you to seamlessly integrate access control for the z/TPF terminal into your existing enterprise authentication methods.

With TPF Operations Server, you can and should restrict access to the prime CRAS terminal because prime CRAS is authorized to issue every command. For non-prime CRAS terminals, you can use file system security support to restrict access to certain commands. File system security support includes a user exit that allows you to specify which users and groups can enter a certain command or use a command with certain parameters. These rules apply not only to ZFILE commands, but to any z/TPF command. For example, you might allow all users to issue a ZDCOR command to display a piece of core memory, but only allow the root (prime CRAS) user to issue ZACOR to alter core memory. These tools provide you with the flexibility to enable operators and coverage programmers with the tools they need to do their job, while minimizing the risk associated with a bad actor or an incorrectly entered command.



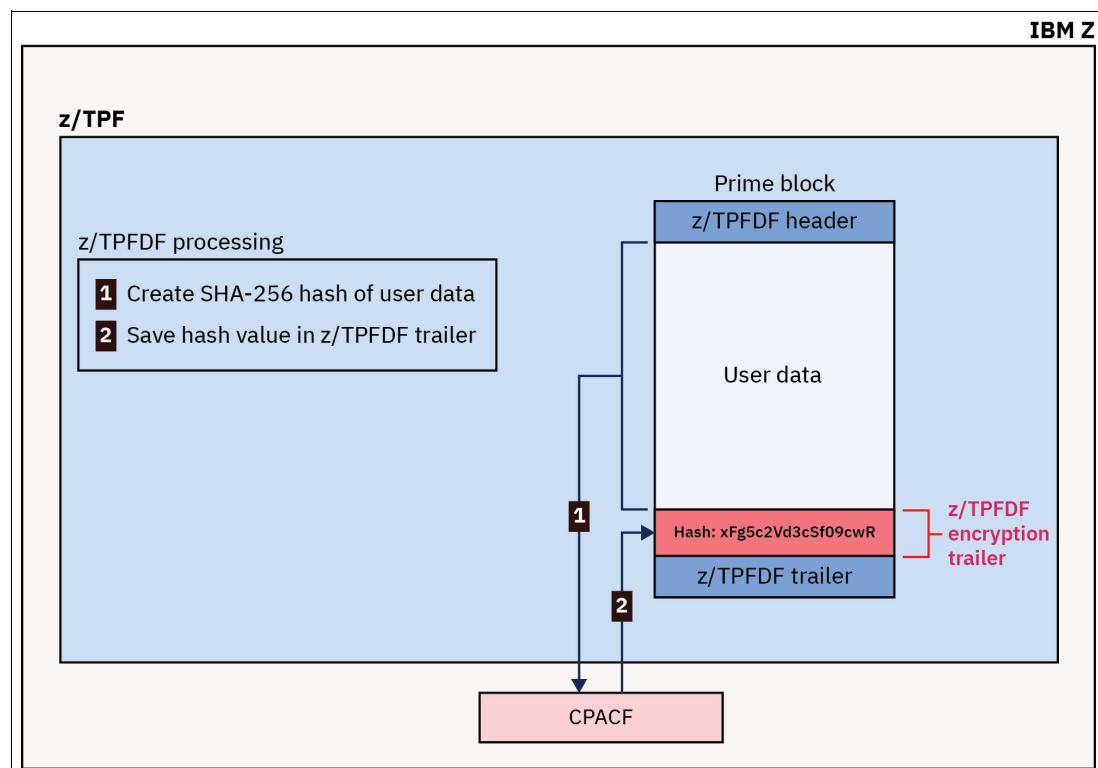
Detecting anomalies and security events

This chapter describes recommendations and highlights z/TPF capabilities and functionality aligned with the Detect pillar of the NIST cybersecurity framework to provide guidance on continuous monitoring of workloads and resources to detect security events. Even if your environment is operating smoothly, it is important to frequently validate your security posture to confirm that you are still compliant with internal and external requirements. For example, a recent code update or configuration setting change might have been made to your system that is using a cipher that is not approved for use or is not using encryption at all such that there is a risk for sensitive data to be exposed.

Despite all the protection mechanisms that exist, incidents can still occur, either accidentally or intentionally. Regardless of the cause, it is important to be able to detect the condition and then have plans and procedures in place on how to react.

6.1 Data corruption detection

A crucial aspect of z/TPFDF encryption is the ability to detect if and when the database contents is not what is expected. For example, corruption of the database from either accidental programming errors or malicious internal actors. You can identify accidental and malicious data corruption using z/TPFDF data integrity verification. This is handled internally by the z/TPFDF database layer by creating a 256-bit SHA digest of the 4K record being encrypted. The 256-bit SHA digest is calculated before the encryption operation takes place. The 32-byte hash value generated by the 256-bit SHA digest operation is saved in the same z/TPFDF encryption trailer where the decryption key name is saved. When the record is read after the data is decrypted, a hash of that data is calculated and then compared to the digest that was saved in the record. If the values do not match, the data is corrupted and is not given to the application program. Verification of the integrity of the data is an optional component of z/TPFDF encryption and can be enabled at any time while the system is processing transactions against the z/TPFDF database. The following set of figures highlight the steps the z/TPFDF database layer takes to ensure the integrity of the data has not been altered either maliciously or accidentally. Figure shows the z/TPFDF database layer creating the message digest hash of the user data and saving the value within the z/TPFDF encryption trailer.



z/TPFDF database layer creating the message digest hash Once the 256-bit SHA digest hash is created, the z/TPFDF database layer encrypts the user data and saves the decryption key name as shown in Figure 6-1.

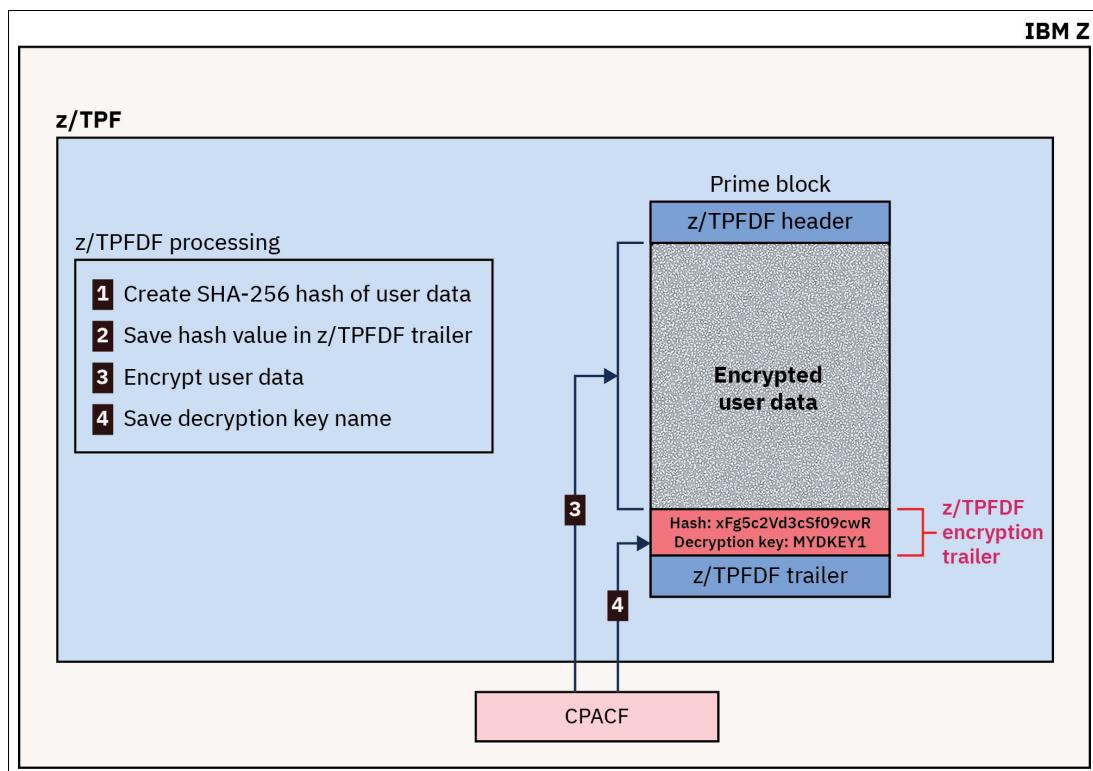


Figure 6-1 z/TPFDF encrypting user data and saving the decryption key name

Now, when this record is read from the database again, the z/TPFDF database layer will first decrypt the user data using the saved decryption key name as shown in Figure 6-2 on page 62.

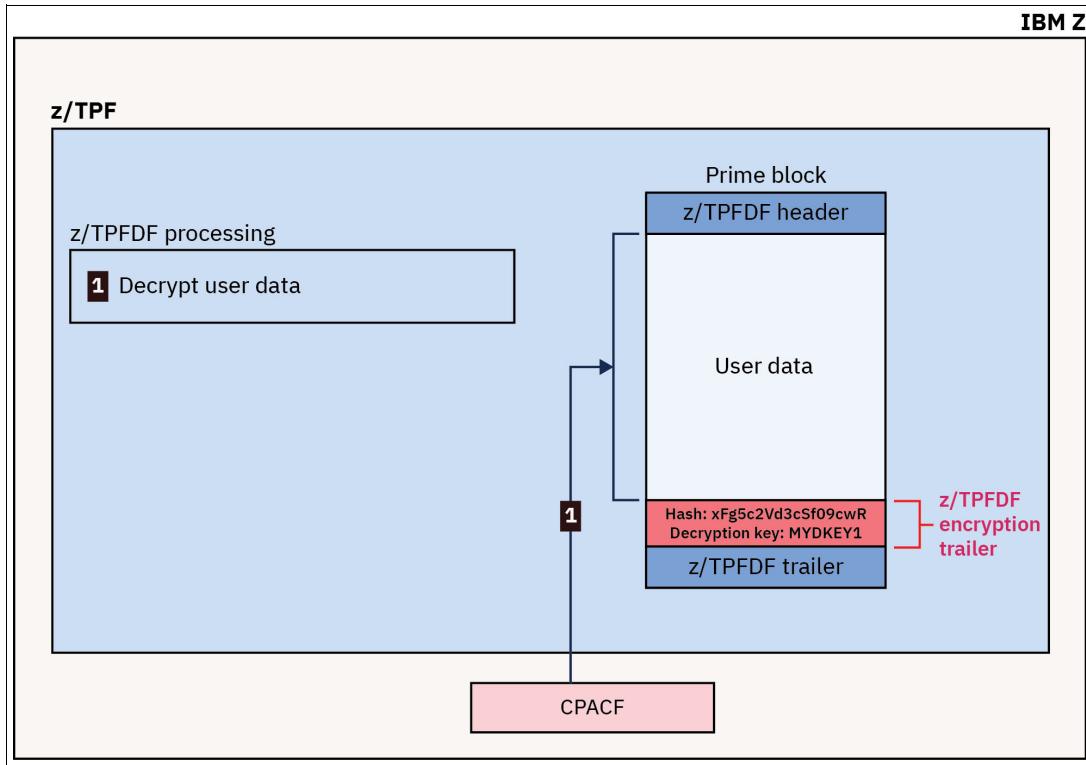


Figure 6-2 Decrypting user data with the saved decryption key name

Now that the user data is in the clear, the 256-bit SHA hash is re-computed and then compared against the saved hash in the z/TPFDF encryption trailer to ensure the hash matches. If it does not match, the data has been modified and will not be passed to the application. If the hash matches, we are sure the integrity of the data has not been compromised. Figure 6-3 on page 63 shows the computation of the SHA-256 hash and the validation of it.

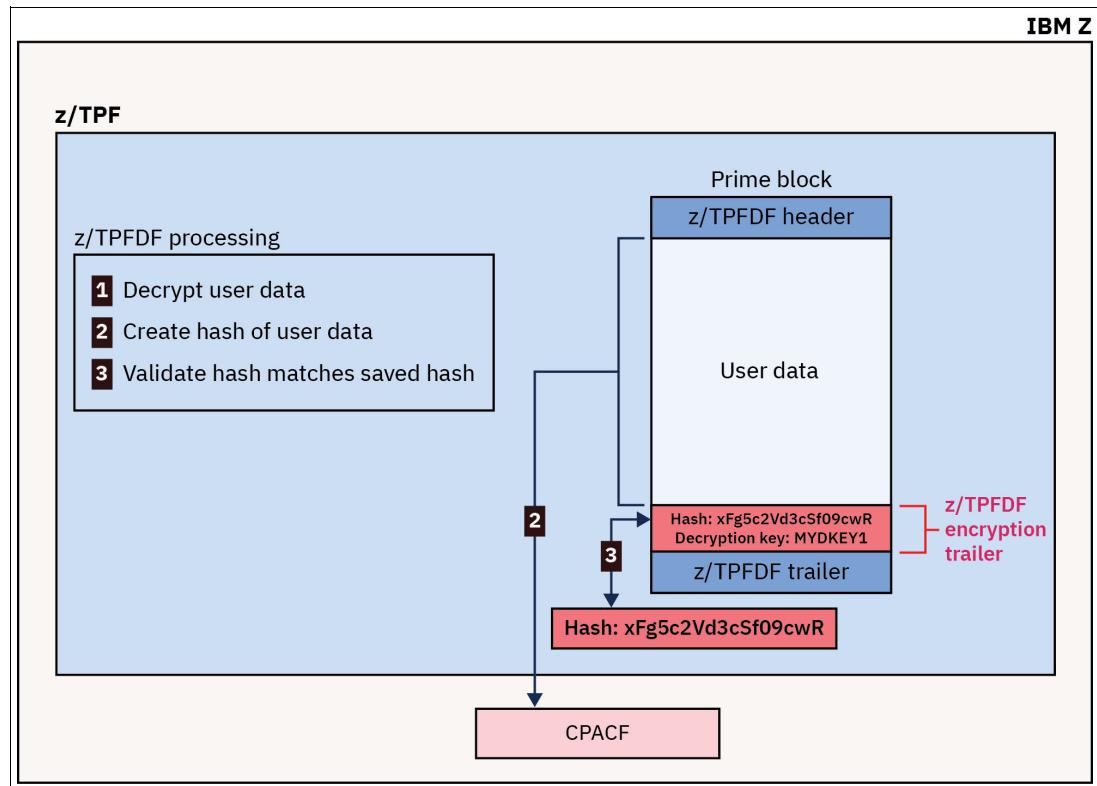


Figure 6-3 Computation and validation of the SHA-256 hash

6.2 Workload overload detection and prevention

You have access controls in place to make sure requests coming into your z/TPF system are from authorized users/sources, and you might even have your API management layer limiting the number of requests for certain services. However, even with that you may encounter situations where the z/TPF system is running low on resources because the total amount of traffic coming into your system is very high. The first steps are identifying the cause of the surge in traffic and which actions to take. Many times, the conditions are not caused by bad behavior or something malicious in nature. For example, your company is having a sale on flights, trains, or hotel stays that result in a much higher booking request rate than normal. Contrast to a programming error that causes a remote application to tight loop sending in many duplicate requests into your z/TPF system. Both examples can overload the z/TPF system, but which actions to take are different. For example, depending on the cause of the overload the appropriate action could be to add more capacity using z/TPF Dynamic CPU support, block one end user, or reject lower priority messages.

A basic workload detection and prevention mechanism is detecting overloads at the protocol layers of TCP/IP. There are two controls that you can put into place on z/TPF to prevent overload situations and they both are controlled through z/TPF network services database. The network services database is the implementation of the standard /etc/services file used by z/TPF, which at its core is a mechanism to assign a name to a server port and protocol combination. The z/TPF system has extended this to allow users to specify characteristics of the server including limits on the number of connections and limits on the number of messages received per second.

6.2.1 Preventing excessive network connections

With the network service database connection limiting support, you can control how many TCP connections a given application can have active concurrently. For example, your secure z/TPF HTTP server is port 443. The connections accessing this port are only received from your API management layer and will at most have 200 sessions active to z/TPF. You can add controls in the network services database to ensure the number of TCP connections established to this port can never exceed some predefined value, for example:

```
tpfhttps 443/tcp weight-100 maxconnin-200 #TPF HTTP Secure Server
```

In this example, the number of inbound connections to port 443 cannot exceed 200. This way, if a rogue user tries to establish hundreds of connections, this control will detect it and prevent that from occurring.

6.2.2 Preventing excessive application traffic

With the network services database traffic limiting support, you can control how many messages per second a specific port (application) can process. This can be controlled at a socket (connection) level and at the overall application level. So, assume there are 200 connections to the HTTP server (port 443). Using the ZIPDB operator command you can interrogate how many messages per second this application port is receiving. For example purposes, assume the maximum number of messages received is 500 messages / sec. You can set up controls in the network services database, for example:

```
tpfhttps 443/tcp weight-100 maxconnin-200 socrate-50 applrate-700 #TPF HTTP  
Secure Server
```

With these settings, each individual socket cannot process more than 50 messages per second, preventing an individual user from maliciously or accidentally flooding the z/TPF system with new messages. The whole application can handle no more than 700 messages per second. Offering a margin for occasional bursts above the 500-limit encountered thus far, it safeguards against and forestalls overload within this application and the z/TPF system.

6.2.3 Preventing excessive service requests

As you expose your services to the outside world, it is important to control the request rate to protect your z/TPF system. This section explains how to prevent excessive service requests at an enterprise-level with centralized API management as well as a more granular level on your z/TPF system with z/TPF REST throttling.

API management rate limits

As discussed in 5.3.2, “Designing and managing access to services” on page 52, an enterprise-wide API management layer such as IBM API Connect® provides centralized access control and management of all services, including those services that are on z/TPF. Figure 6-4 shows a centralized API management layer or API gateway controlling access to the services in your enterprise from your cloud providers.

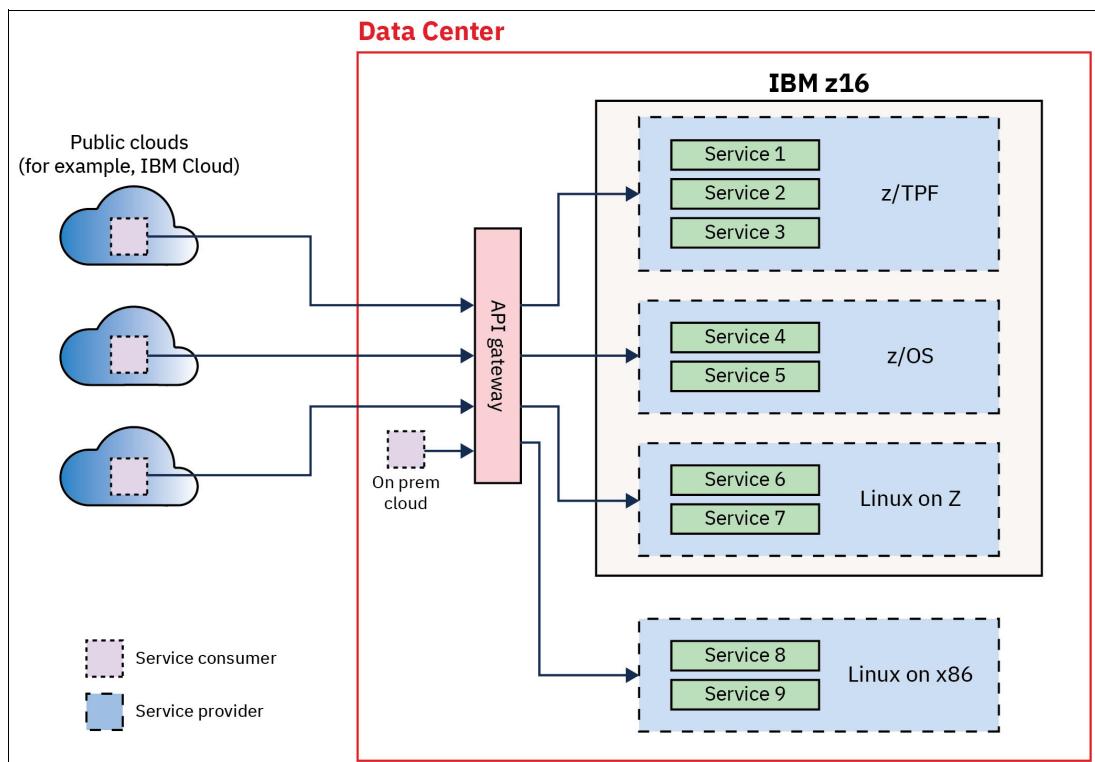


Figure 6-4 Centralized APIM for all your services

One of the many benefits of having a centralized API management layer is the ability to apply rate limits to your APIs. Rate limits are an important mechanism to control the rate of incoming requests to your backend services, protecting them from becoming overwhelmed. There are different types of rate limits that can be defined within your API management layer, and they are very much vendor dependent. For example, you can control how many API calls are processed for a given service across a specified period, say one second.

So, if the number of API requests received reaches your defined limit, warnings can be generated or requests can be rejected. You can also get even more granular by defining limits for a given remote user. For example, if the number of requests from a specific remote user reaches 100 API calls in a second, then subsequent requests from the user will be rejected for the remainder of that 1-second interval. Using your centralized API management layer as a first line of defense for your backend systems is key to detecting and responding to traffic anomalies, whether they are malicious or not.

You might set rate limits for certain transaction types that are non-revenue generating (for example, read reservation record transactions) or very resource intensive (for example, power shopping), but not for transaction types that make you money (for example, bookings). When you have dozens to hundreds of services, the rate of any one individual service might not be high, but the sum of requests across all services might cause an overload condition.

While controlling the rate of requests at your centralized API management layer allows you to set up rate limits for all the APIs across your enterprise, in certain cases that may not be enough. If all your APIs do not go through your API management layer, or you need even more granular control, you may need additional layers of protection. You can set up limits on concurrent APIs within the z/TPF system itself using the z/TPF REST throttling support.

6.2.4 z/TPF REST throttling

The z/TPF REST throttling support allows you to limit the number of REST API requests on an API basis allowing for greater system stability. On z/TPF you can control a given REST API by limiting the number of concurrent requests being processed. This type of control prevents a short sudden burst of inbound requests that your API management layer is not granular enough to handle. For example, say the API management layer allows for 1,000 API requests per second to be processed. Across a full second, the system can easily handle 1,000 requests. However, what happens if 500 of those requests all arrive to the system within the 1/20th of a second (50 milliseconds). Processing 500 requests of this API concurrently on z/TPF may be considered overload, while the API management layer allowed this as most of the time control is at a second basis. With REST throttling, another layer of protection can be added to prevent scenarios like this.

Figure 6-5 on page 67 shows three API services on your z/TPF system. There is a service policy defined for each with service 1 allowing 50 concurrent requests, service 2 is allowing 10 concurrent requests and service 3 has an unlimited number of concurrent requests. Since Service 2 has reached the number of concurrent requests active in the z/TPF system when a new request comes in for service 2, that new request is rejected by the z/TPF REST stack and never makes it to the z/TPF application that provides that service.

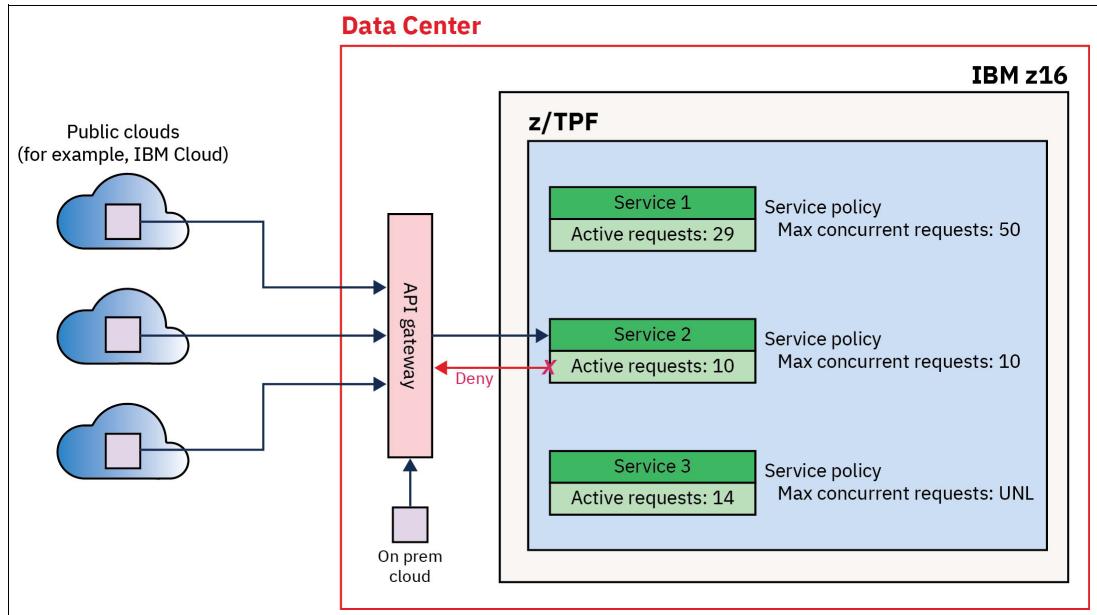


Figure 6-5 REST throttling example for concurrent requests

While controlling access based on the number of maximum concurrent requests can help control the load and impact of a given service, there may be cases where you want to limit a given REST service if the system is constrained by resources like ECBs or CPU capacity. In these cases, you may want to shed work coming into the system by limiting individual services. Say, for example, that you have various types of services, some related to booking new passengers, while other services are just looking at seat maps or availability on flights. You might configure non-critical REST services (for example, customers looking for seat maps or availability) to reject requests if the system is at 90% CPU utilization or higher.

Figure 6-6 on page 68 shows services in z/TPF with a priority class defined to them. Assume that the LODIC_USRLOD1 has a utilization shutdown of 75%. In this example, there are only three Service 2 APIs concurrently being processed, but the system has spiked to an overall utilization of 80%. Requests destined for Service 2 will be rejected and not processed until the CPU utilization drops back below 75%.

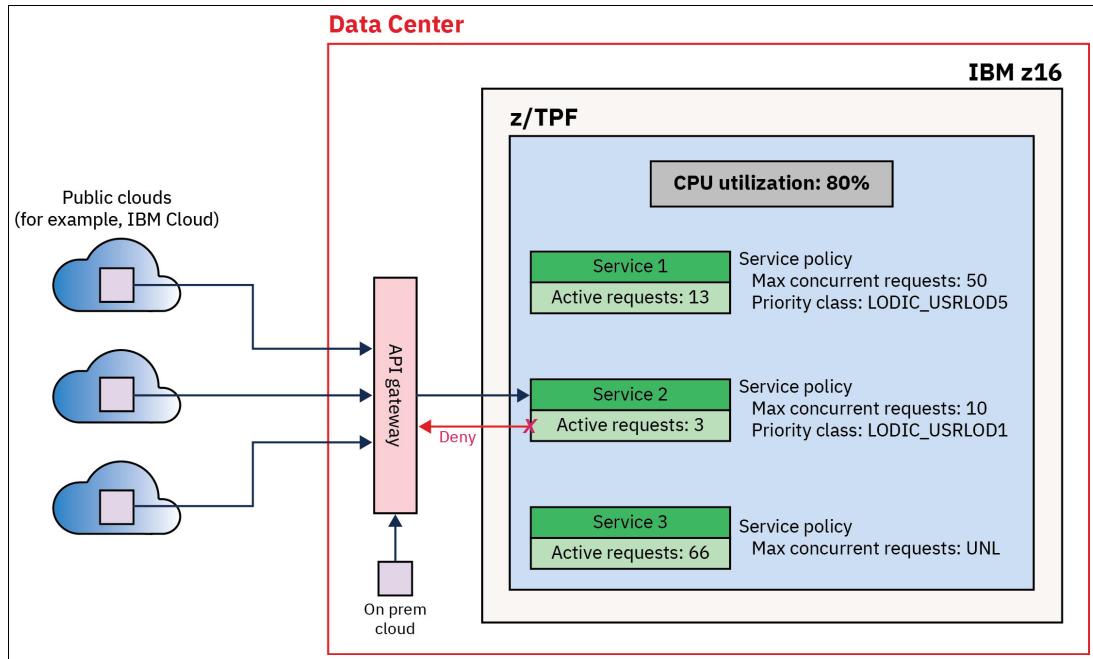


Figure 6-6 REST throttling example using LODIC class

6.2.5 Preventing system overload

In addition to detecting and responding to workload overload to prevent new requests for a given REST service to begin processing under certain conditions, the z/TPF system has built in overload protection for all work coming into z/TPF. Each CPU (I-stream) on the z/TPF system has work lists associated with it. Some of the notable ones are a cross list that allows work to be moved between CPUs, a ready list for in flight work that is ready to continue processing, and an input list which is used for new work that has arrived from the network. The processing of these lists is top down, meaning the cross list is processed first, followed by the ready list, and then the input list (new work). Before the z/TPF system checks the input list, the previous lists (cross and ready) must be empty. Figure 6-7 on page 69 shows a z/TPF system with three CPUs and the associated lists for the work on the system.

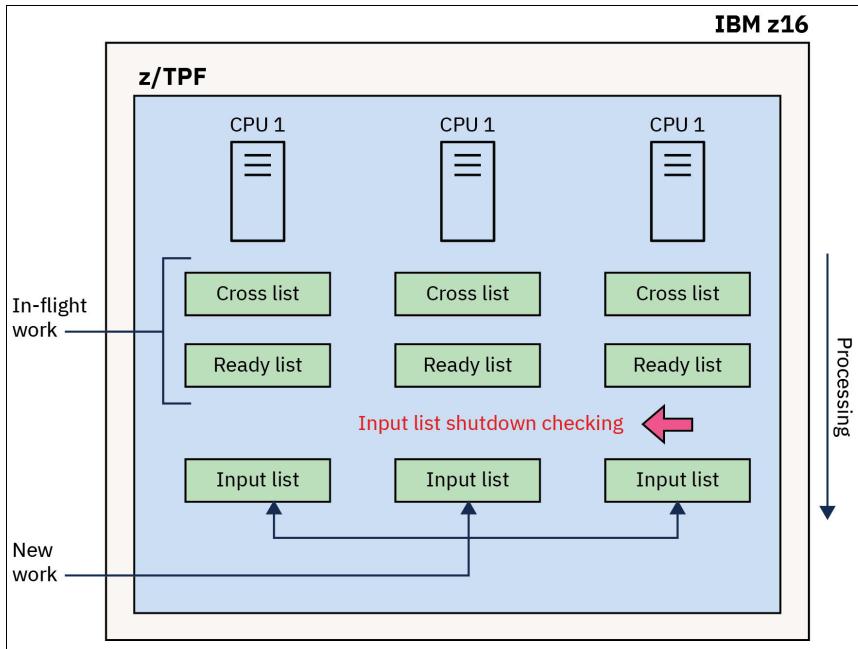


Figure 6-7 z/TPF task dispatcher

A system-wide safeguard z/TPF has is that the input list will not be processed if the system is running low on resources. This is referred to as input list shutdown, which requires enough in-flight work to finish processing to free up enough system resources where it is safe to start processing new work again. The shutdown levels are customer configurable, and largely depend on your overall resource allocations. For example, say you have 3,000 ECBs defined to your z/TPF system and set an input list shutdown level for ECBs to 2,700. Once the system has 2,700 or more active ECBs in the system, the input list will no longer be processed until the level of in use ECBs drops below the shutdown level. This mechanism allows for greater system stability and allows the system to put off starting to process new work until the in-progress work already in the system is reduced.

The system has an input list shutdown monitor where z/TPF operators are notified when shutdown levels are reached. In addition, the resource usage can be continuously monitored using z/TPF specific tooling like continuous data collection (CDC) or real-time RTMC. Detecting resource overloads is important and allows the customer to automatically shed or defer less important work. If there are frequent instances of input list shutdown, that can be a sign that you need to add more resources to your z/TPF system to handle higher workload volumes.

6.2.6 Monitoring network anomalies

Another detection mechanism for workload overload situations that the z/TPF system has is the socket monitor, which detects numerous conditions allowing you to act on including:

- ▶ Output messages are not sent for extended periods of time because the remote node is not permitting z/TPF to send any data (TCP send window blocked condition). There is a problem with the remote node that needs to be investigated.
- ▶ One or more input messages have been queued to a socket for an extended period, but no z/TPF application has read those messages. This indicates an issue with a z/TPF application.

- ▶ Backlog limit exceeded for a z/TPF TCP server application. Connection requests from remote clients are coming into z/TPF faster than they can be processed by the server application and have reached the limit defined for this server application. Therefore, some connection requests are rejected.
- ▶ The number of ECBs queued up waiting to send on one socket has reached a threshold. This often means multiple z/TPF application instances are using the same socket and generating data much faster than the data can be transmitted to and processed by the remote application.

The socket monitor detects several different networking conditions and allows the user to act on that condition. Take the situation where we have a z/TPF socket or TLS session used to send data to remote Linux system. On z/TPF, multiple transactional ECBs send data across that one session as part of the transaction. If the application on the remote system is having problems and stops reading from the session, the remote socket's receive buffer will fill with data, causing the remote TCP/IP stack to tell z/TPF to not send any more data at this point. As z/TPF transactional ECBs continue to issue APIs to try and send data on this session, eventually the socket send buffer on z/TPF becomes full which means subsequent send APIs becomes blocked, causing transactional ECBs to become suspended. The number of ECBs queued waiting to send on this socket will continue to grow until the remote application starts reading from the socket again (freeing up space in the remote socket receive buffer, which in turn causes the remote TCP/IP stack to inform z/TPF that it is fine to send more data now), or the send API times out (assuming the application set a timeout for this socket). Figure 6-8 shows a remote system that is no longer responding, the remote's receive buffer, z/TPF send buffer full, and z/TPF transactional ECBs queuing waiting to send.

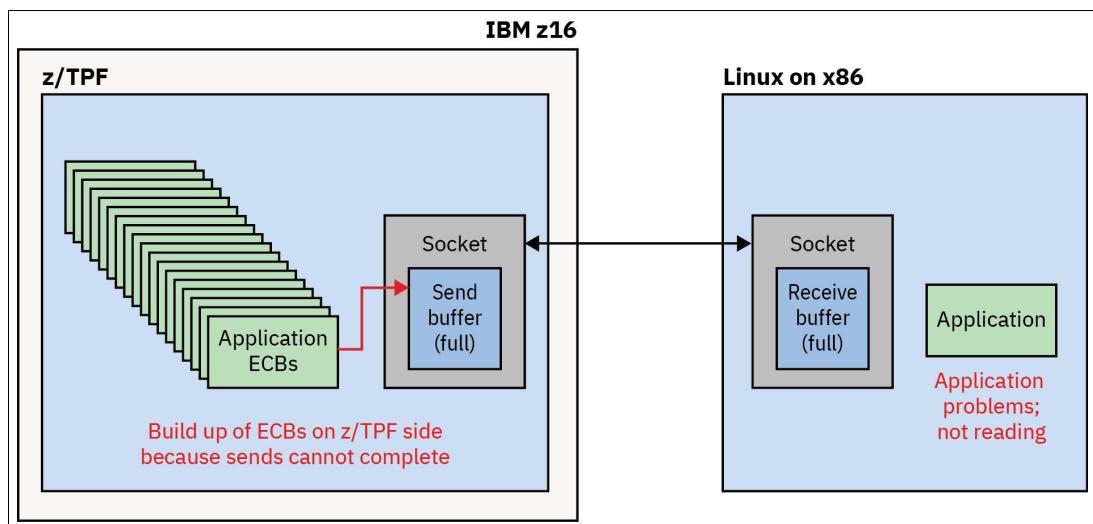


Figure 6-8 Socket monitor - remote application is not reading data

Once the number of ECBs waiting to send on a session reaches a certain threshold, the socket monitor detects this condition and passes control to a user exit to allow you to act. In this example, the appropriate action is to close the session preventing more ECBs from becoming queued. If that is allowed to happen it could eventually drive z/TPF into an input list shutdown condition due to low resources or running the z/TPF system out of resources completely. Figure 6-9 on page 71 depicts the z/TPF system entering the socket monitor user exit, and the customer's user exit code choosing to close the session. This would result in every queued ECB waiting to send to get an error return on the send API issued and allow those ECBs to clean up and exit.

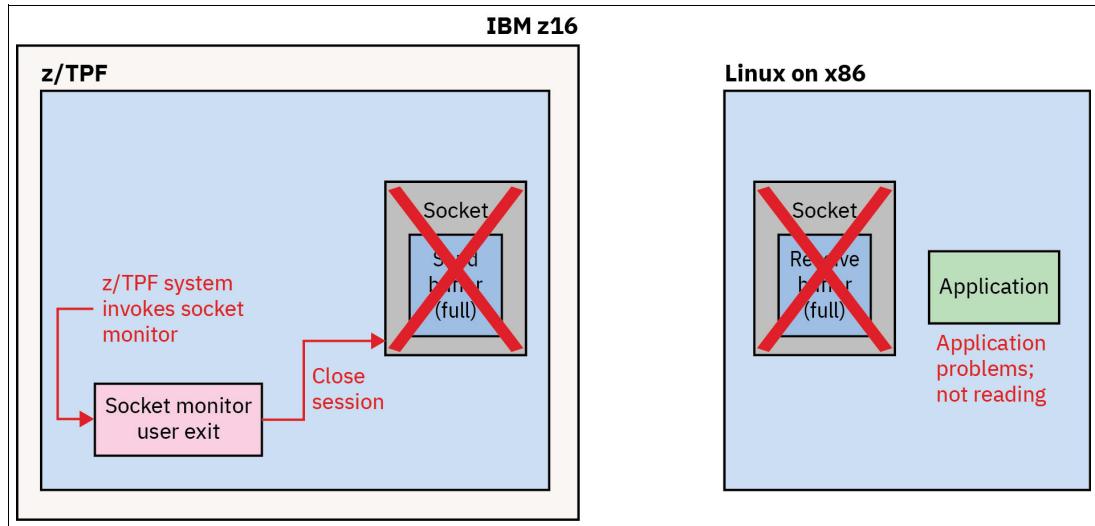


Figure 6-9 *Socket monitor - clean up problematic sessions*

6.3 Resource overload detection and prevention

We have discussed the ability to limit the amount of traffic received on your z/TPF system, but how can you prevent a single or a few malicious requests from consuming too many resources? From a z/TPF point of view, this is monitoring specific ECBs on the system to ensure they are not consuming resources abnormally.

6.3.1 Monitoring z/TPF ECB resources

The z/TPF ECB resource monitor provides a centralized facility that monitors the use of selected system resources for each ECB. The ECB resource monitor detects and, optionally, stops an ECB that requests excessive amounts of monitored resources. The z/TPF ECB resource monitor allows you to monitor single ECBs or a group of ECBs that are part of the same type of work being performed. You can set limits on a variety of resources, for example, how much system storage is obtained (like system work blocks), database accesses, ECBs created, system errors and messages sent to the console. This type of control allows you to prevent malicious attacks as well as application program logic or coding errors that can cause system outages.

The z/TPF ECB resource monitor allows two levels of detection. When the first level is reached, a system error or warning message can be issued, and control is returned to the application. When the second level is reached, the ECB can be forced to exit with a system error.

For example, say you are monitoring ECBs associated with flight availability. You can set up controls to limit the number of DASD find operations to say 500. Due to an application program error, instead of checking flights for just the data specified on the search request, the application is looking at every flight that month causing a single ECB to attempt to access thousands of records. The ECB resource monitor can warn you of this occurrence and optionally target the offending ECB to abnormally exit with a system error.

6.3.2 Monitoring resources using Runtime Metrics Collection

Another way to monitor system resources and usage is to use Runtime Metrics Collection (RTMC). RTMC is a flexible and extensible monitoring and analytics framework that can be used to identify performance anomalies and the likely cause to know how you should respond. Sample causes include:

- ▶ An unexpected surge in valid traffic. If this is revenue generating traffic and the system is low on capacity, you might choose to add capacity on the fly using z/TPF Dynamic CPU support.
- ▶ An application on z/TPF is consuming many more resources per transaction than it used to. If a new version of the application was recently loaded that should not be consuming significantly more resources, you can back out that code load.
- ▶ Messages from one remote end user or business partner is consuming many more resources per transaction than it used to. If this change in behavior is unacceptable, you could contact that partner to see if they made a recent change on their side and if so, ask them to back out that change.
- ▶ A higher-than-normal number of messages received are being rejected by z/TPF because those messages are not formatted properly. This might be a denial of service (DoS) attack or just a remote application bug. Either way, you might choose to update your APIM layer, firewalls, or both, to block traffic from that source.

As you can see, sometimes the cause is organic transaction growth, sometimes the cause is an application code bug, and sometimes the cause is because of a security related incident. RTMC allows you to detect these conditions in real-time and help classify what type of problem is occurring on your z/TPF system to mitigate the problem.

6.4 AI-infused detection

Regardless of whether you are using the TPF Operations Server or another console automation product, you should use ML and AI to monitor and analyze the console log to detect anomalies. For example, prior to this month operator X never displayed more than 10 database records in a week, and no operator displayed more than 25, but now operator X is displaying more than 100 database records per day. Why? Another example is where for the past several months operator Y has worked the day shift, but this week in addition to the day shift, operator Y is logging on for 20-30 minutes at night and issuing commands operator Y rarely or never has issued in the past. Perhaps operator Y is up to no good, or maybe operator Y's credentials have been stolen and are being used by a bad actor in addition to by operator Y. These are just two or many examples of potentially suspicious behavior that needs to be investigated.

Section 6.3.2, “Monitoring resources using Runtime Metrics Collection” on page 72 explains how RTMC can analyze transactional and system data to help quickly pinpoint the cause of abnormal system resource consumption, but you can also add AI to use the data RTMC provides to detect suspicious transaction patterns over time that may not cause system resource issues. For example, why over a period of six hours is one end user buying many seats on flights or trains between city A and city B for the same day? Why is one end user making multiple transactions today using loyalty points, then quickly canceling each one?

In addition to detecting bad end user behavior, AI can also use RTMC data to detect business impacts caused by an application logic error or bad data load. For example, in recent months the average price paid for an airline ticket booked well in advance from city A to city B was \$800.

If there was a bad data load resulting in a massive drop in price, like from \$800 to \$80, the rate of tickets sold in that market would likely increase significantly where that would be a trigger for AI that there is potentially a problem. However, if the bad data load had resulted in the average ticket cost being \$700 instead of \$800, the volume of tickets sold might not change to trigger an alert, but AI can detect the problem by analyzing the transactional data such that you can quickly take corrective action to minimize the loss of revenue caused by the bad data load.



Responding and recovering from security events

This chapter describes recommendations and highlights z/TPF capabilities and functionality aligned with the Respond and Recover pillars to ensure appropriate action, business continuity, and timely recovery after a cybersecurity incident. In many cases, the speed and quality of your company's response and recovery from an incident depends on the level of preparation, which begins with incorporating security by design in your software development processes (See Chapter 2, "Security by design with risk management frameworks" on page 5) as well as establishing operational playbooks and procedures in case of a security event.

In addition to having a plan, it is important to practice your security incident response and recovery processes, using your existing procedures and playbooks. For example, if your primary data center suddenly became unusable and you had to move to a DR site, that is not the ideal time to discover that a step in the script does not work, or that there is a glaring omission in the script. Practicing and validating the recovery procedures is not a one-time activity, it needs to be done periodically. For example, you do not want to find out the hard way during an incident that your playbook contains an old/wrong IP address or hostname, or that when the playbook was created there were three systems involved in processing a workload, but now there are four and that 4th system is not accounted for in the playbook.

Many organizations have had DR sites and procedures for decades, but that is not the only recovery plan you need. In today's world, you need to have a plan for how to recover from a cyberattack or other incident that results in significant data corruption, how to recover from bad code being loaded to your system, and how to handle the case where a flaw is identified in one of the protection mechanisms currently in use (like a specific encryption cipher).

7.1 Security incident response using cryptographic agility

Cryptographic agility, or the ability to quickly identify and change the usage of a given key or cipher, is of utmost importance in responding to and recovering from security incidents. z/TPF provides cryptographic agility by allowing you to control cryptographic usage through configuration (z/TPF operator commands and TLS configuration files), to quickly and easily change keys and ciphers used by the z/TPF system and your applications.

In the case that the industry deems that a cipher is no longer safe to use, it is necessary to quickly identify exactly which applications, databases, and secure connections are configured using that specific cipher. There are operator commands described in section 4.3, “Compliance reporting on z/TPF” on page 13 that generate reports to determine that information. If you are exposed (are using the cipher in question), take the following actions to switch to use a different (approved) cipher:

- ▶ For secure connections using the cipher in question, update the TLS configuration files as described in section “TLS configuration changes using cryptographic agility” on page 32 to remove that bad cipher and, if necessary, add additional cipher(s) that are approved for use.
- ▶ For a z/TPFDF database that is encrypted using the cipher in question, create a new secure key using an approved cipher, assign that key to the z/TPFDF database, and rerun the CRUISE utility as described in section “z/TPFDF database encryption” on page 42 to re-encrypt all records in this database using the new key.
- ▶ For applications directly using a z/TPF secure key that is using the cipher in question, follow the steps for rotating keys described in section “Data encryption within your application” on page 38.

If there is a requirement from your organization to upgrade to a newer and stronger cipher, then the same actions described above can be taken. Although not as time critical, you may still need to react quickly to adhere to the new requirements.

7.2 Security incident response with access controls

In many incidents, the threat to your z/TPF system is coming from a remote user or application. These types of incidents can be accidental or malicious, but the response is generally the same. Say you are continuously monitoring your z/TPF system using RTMC, and you detect an anomaly that results in a large increase in CPU utilization. You identify a particular user or remote application as the cause of the increased resource usage. In many cases, this is a defective remote application program, for example looping sending in requests to z/TPF. However, this may also be a malicious user attempting a denial of service (DoS) attack on the z/TPF system.

When something like this occurs, procedures should be in place to quickly disable access to your z/TPF system for a particular user or remote system. This is generally done in your API management layer to disable individual users, or within firewalls that are in place to protect your environment to disable access from specific remote systems or to specific ports. Firewalls controlling access to your z/TPF system can reside externally, or on z/TPF by using the z/TPF IP packet filtering capability.

In both cases, a procedure to block a specific remote user or system from accessing your z/TPF system should be created and practiced periodically to ensure that when an event like this occurs you can quickly respond.

7.3 Data corruption recovery

As mentioned in section 6.1, “Data corruption detection” on page 60, z/TPFDF data integrity verification can detect data corruption in your database using an internally managed 256-bit SHA digest. If z/TPFDF detects corruption in even a single record, meaning the generated hash does not match the saved hash, z/TPFDF returns an error to the application. Additionally, z/TPFDF issues a system error to the z/TPF console, alerting your operations and coverage staff that data corruption has been detected and allowing them to immediately act.

Whether data corruption is the result of a cyberattack, a programming error, or human error, you need to be able to restore your data to a known, good state. As mentioned in section 5.2.4, “Protecting your data from corruption” on page 48, corrupted production data is mirrored to other active z/TPF systems and disaster recovery sites within seconds. To protect your data against corruption, best practice is to maintain multiple, point-in-time, cyber resilient backups of your production data using a solution like IBM Safeguarded Copy.

If data corruption does occur, you need to restore your data to the most recent, uncorrupted state. Because you have multiple backups made at different points in time, one of the first questions is determining which backup contains the most recent uncorrupted copy of your production data. In some scenarios, like a bad program load or human error that is quickly recognized as being the cause of the corruption, it might be easy to determine when the corruption occurred. In this case, you can quickly determine which backup to restore.

Figure 7-1 shows source volumes, H1, for a production system and backups were made at times t1-t4 using IBM Safeguarded Copy and secured in the IBM Safeguarded Copy Backup Capacity. If the event that corrupted data occurred after time t3 but before time t4, the copy at time t3 would contain the most recent copy of uncorrupted data. This means corrupted data exists in H1 and in the vault at time t4. To restore the most recent backup with uncorrupted data, you would restore the copy from time t3 to recovery volumes R1.

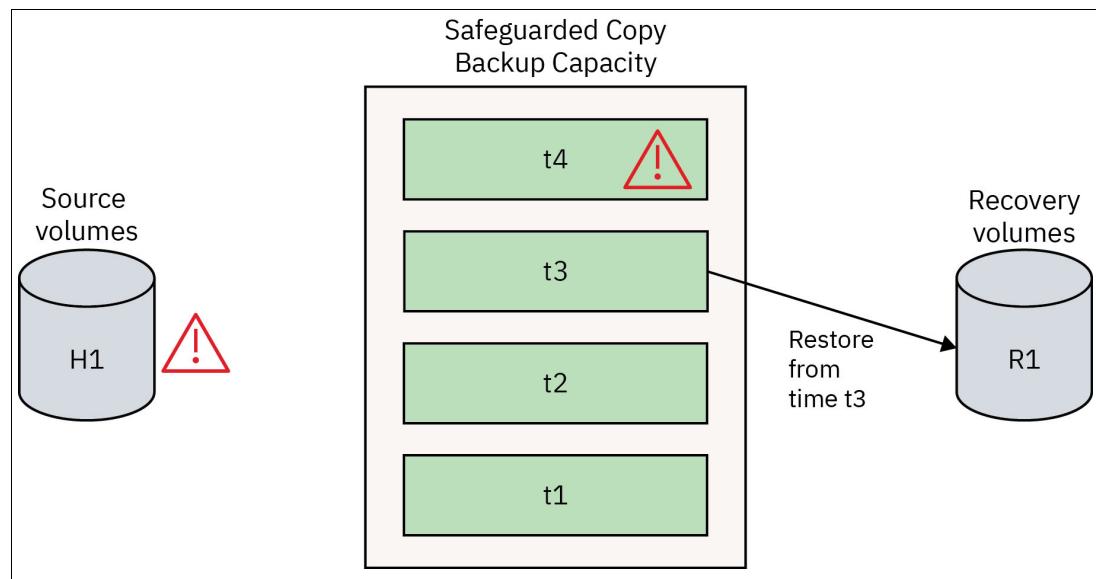


Figure 7-1 Creating recovery volumes from a secure backup

After the restore to R1 is complete, you need to validate that you have an uncorrupted copy of your data. Data validation should be done even if you are confident that you identified the event that caused corruption and have selected a backup that was made prior to the event. Validating data includes Initial Program Loading (IPLing) a z/TPF system using R1 and using z/TPF utilities or customer utilities to validate the database structure and contents. If some of the data is still corrupted, you would need to restore from an older backup and repeat the validation.

In cases where the time the corruption occurred can't be quickly or easily identified, you might need to simply start the restore and validate process with the most recent backup. In the case of Figure 7-2 on page 79, that would mean restoring and then validating the backup from time t4. If that copy contains corrupted data, you restore and validate the next backup, repeating that process until a backup is found with no corrupted data or the least amount of corrupted data.

After the most recent good backup is identified, there are two approaches to restoring the data depending on the extent of the corruption. If the corruption is isolated to a few records or a single database, you could extract uncorrupted copies of the records or database from the recovery volumes R1 and restore them to the production source volumes. For individual records, z/TPF commands like **ZCFIL SAVE** and **ZCFIL RESTORE** could be used to save records from R1 to a file or general tape and then restore from the file or general tape to the H1 production volumes. For z/TPFDF databases, the CRUISE utility could be used to save individual subfiles or entire databases from R1 to tape and then restore those subfiles or databases from tape to the H1 production volumes.

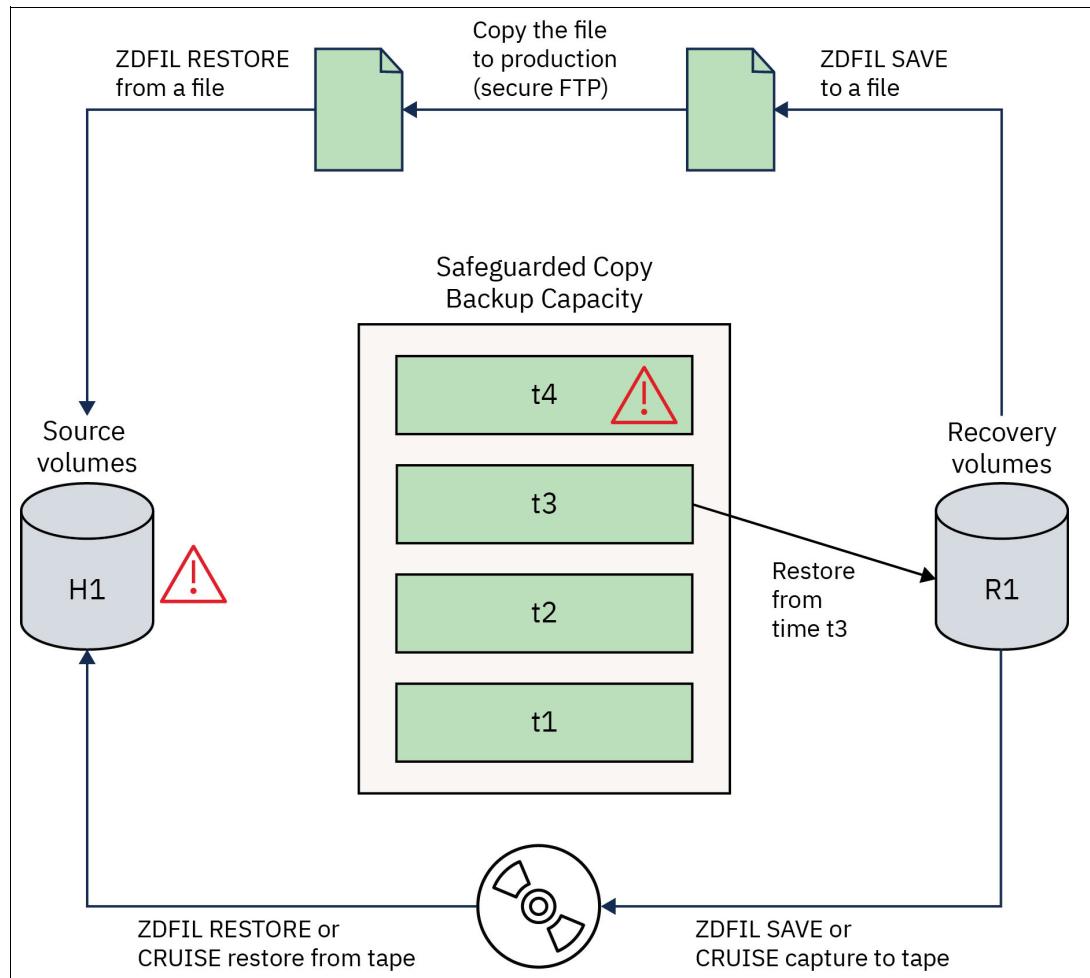


Figure 7-2 Starting the restore and validate process with the most recent backup

In the case of extensive data corruption or corruption that involves multiple related databases, it might be faster and easier to use the entire set of R1 recovery volumes for your production system. Depending on your requirements and system configurations, you might have several options when it comes to recovering your production system.

One option is to use the validated R1 recovery volumes for your production system, which might help reduce the recovery time objective (RTO). A z/TPF system was already IPLed using the R1 recovery volumes to validate the data, so it should still be running. To make it the new production system, several actions might still have to be taken, like routing the network to this system and bringing up other supporting systems. Note that copy services, including continuous mirroring to a disaster recovery site and point-in-time copies to protect against data corruption either need to be established with the R1 recovery volumes as the source or you need to plan to move the production system back to its original location.

A second option is to copy the R1 recovery volumes back to your production source volumes or to a disaster recovery site. As shown in Figure 7-3, copy services like IBM Global Copy would be used to copy the R1 recovery volumes back to the source volumes. After the copy is complete, the production system can be IPLed and production workload can be restarted. In this case, the copy service relationships (continuous mirroring and point-in-time copies) still exist for the H1 source volumes and should resume normal processing.

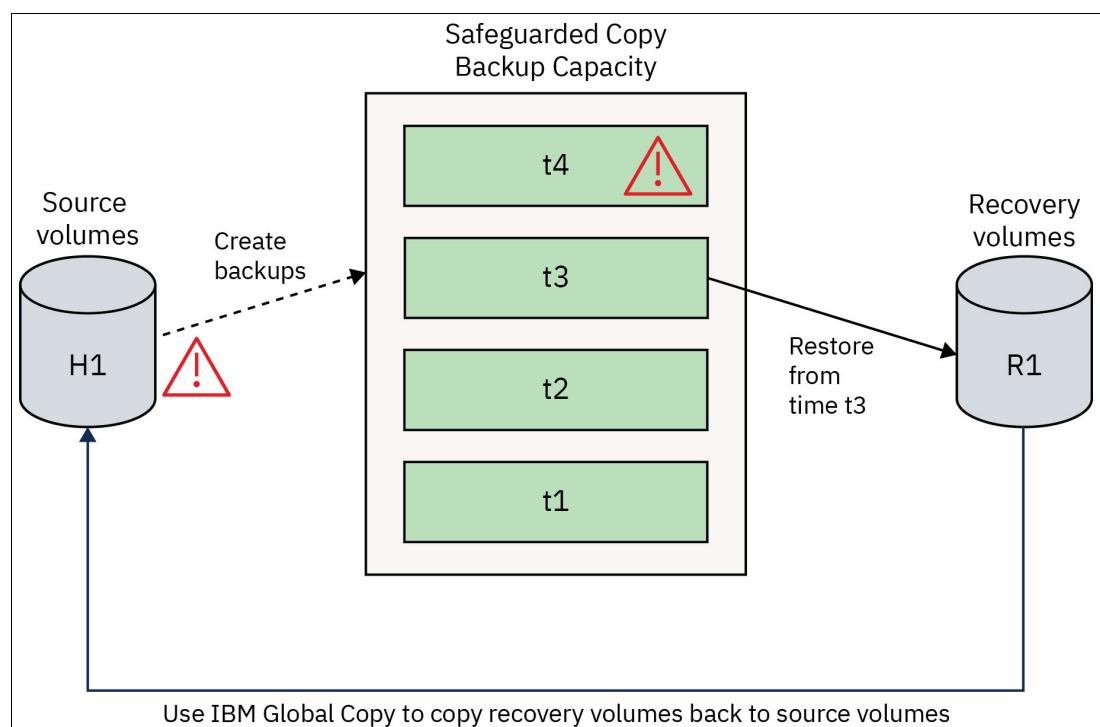


Figure 7-3 Copying recovery volumes back to your production source volumes

Note that z/TPF systems maintain two copies of data on two separate volumes (prime and duplicate modules) for high availability purposes. To ensure a consistent copy of the production data, backups are created using only one copy of the data, usually just the prime modules or just the duplicate modules. As a result, restoring a backup from the IBM SGBC to the R1 volumes restores just the prime modules or just the duplicate modules. To restore both prime and duplicate modules, two options are available. First, after the system is restored and IPLed on a single set of volumes, the z/TPF **ZMCPY** command can be used to recreate the second set of volumes.

A second option is to use copy services like IBM Global Copy to copy the R1 recovery volumes to two sets of source volumes - one set representing the prime modules and a second set representing the duplicate modules. After the copies are complete and the volume serial numbers (VSNs) updated to match the prime and duplicate module VSN requirements, the system can be IPLed using a full set of volumes.

Because IBM Safeguarded Copy backups are point-in-time copies and created periodically throughout the day, the backup used for the restore might have been created several hours before the corruption occurred. For example, if the last uncorrupted backup was taken at 10am and data corruption occurred at 2pm, the system would be restored using the 10am backup. However, there were 4 hours (10am to 2pm) of database updates that are not part of the IBM Safeguarded Copy backup.

To restore data up to the point of corruption, z/TPF record logging can be used. The z/TPF record logging facility captures database record updates to tape and best practice is to capture all record updates except for records that contain transient or temporary work data. In the case of database corruption, the z/TPF **ZFRST LOG** command can be used to restore logging records for a specific period. In the previous example, the IBM Safeguarded Copy backup restored the database back to 10am. Using the z/TPF record logging facility, you can restore the database back to 1:59pm, a minute before the corruption occurred, and minimize data loss from the event.

7.4 Problematic application and system code recovery

Despite best efforts at testing and validating code before loading it to your production system, a defect might surface only after being loaded to production. Some defects, such as an incorrect display, have a minimal impact to your business. However, severe defects, like a defect that impacts the availability of services or corrupts data, require immediate action.

As mentioned in section 5.1.2, “Securely installing software and firmware” on page 18, the z/TPF program loader is used to securely load both z/TPF system and application programs to the z/TPF system. In the case of a defective code load, the z/TPF program loader allows you to quickly revert to a prior version of programs without having to locate those versions or perform additional code loads.

The methods used to revert to prior programs depends on how those programs were loaded. The z/TPF program loader provides two ways to load code: the image loader and the dynamic program loader.

The image loader can be used to load both z/TPF system and application programs to your z/TPF system, including the z/TPF control program or kernel and associated components. The image loader supports up to 8 separate images, where each image is a complete set of z/TPF system and application programs.

An active z/TPF system runs the programs from one image (image A in Figure 7-4 on page 82), and the image loader can load programs to any of the other images. The example z/TPF system in Figure 7-4 on page 82 is currently running on image A and the z/TPF image loader is used to load programs to image B.

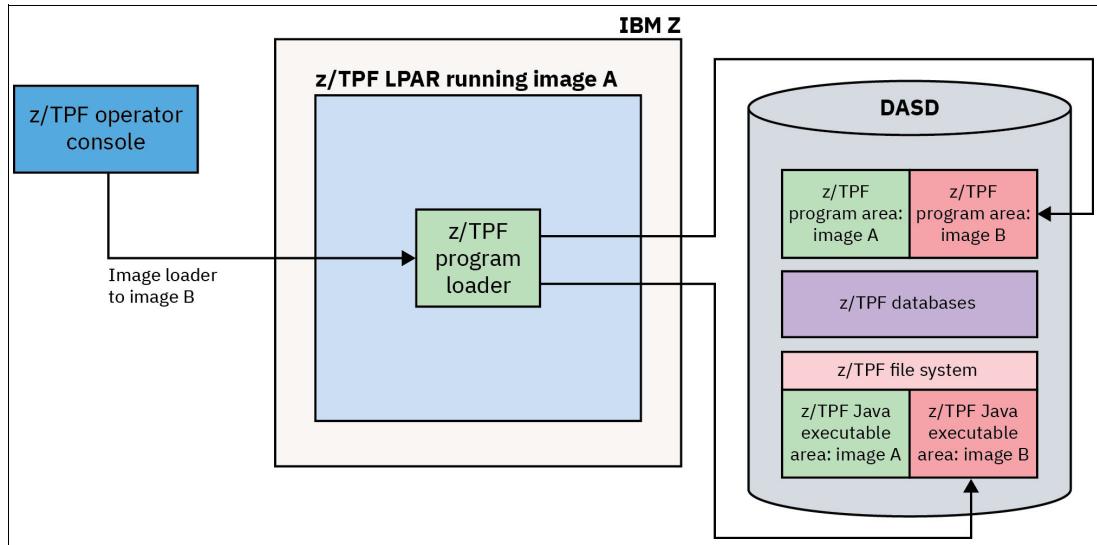


Figure 7-4 Loading z/TPF code

To use the programs in another image, the z/TPF system must be IPLed (rebooted) using the other image (image B in Figure 7-4). If there are any issues with the programs on the new image, you can quickly return to a prior image (back to image A) by IPLing the z/TPF system using that image. Because multiple images are available, you can easily and quickly switch between images.

The dynamic program loader can be used to load application programs and most z/TPF system programs to your z/TPF system (notable exceptions are the z/TPF control program, its associated components, and a small number of z/TPF system programs). The main advantage of using the dynamic program loader is that there is no disruption to traffic, meaning no need to stop and restart application services. This allows you to load new code whenever you need to without impacting availability.

With the dynamic program loader, programs are packaged in a loadset and represent new versions on top of the programs in the current image. Loadsets can contain any combination of assembly, C/C++, and Java programs and can even include new programs that are not part of the current image. Figure 7-5 on page 83 shows deploying a loadset called *TPFLLOAD1* to the current image, image A.

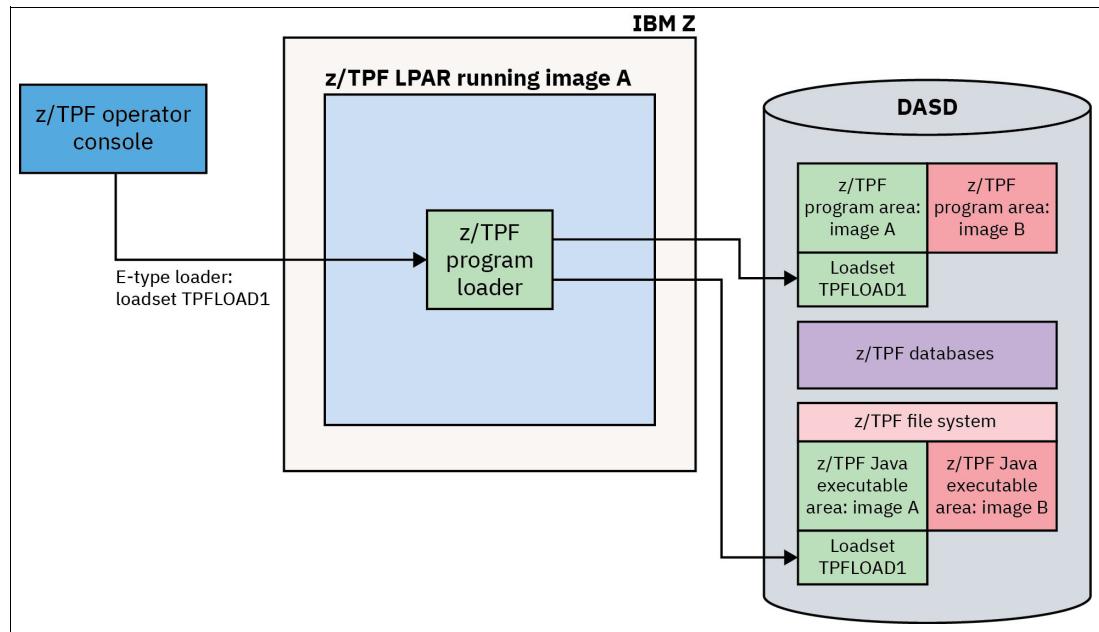


Figure 7-5 Deploying loadset TPFLLOAD1 to the current image

After the load is complete, the z/TPF operator issues a separate command to activate the loadset, which instantly makes the programs in the loadset available to new transactions. For example, version 1 of program ABCD is in the Image A program area and version 2 of ABCD is in loadset TPFLLOAD1. After TPFLLOAD1 is activated, new transactions will use version 2 of program ABCD.

If there is an issue after activating a loadset, the z/TPF operator can quickly deactivate that loadset so new transactions will not use any programs in that loadset. In this example, after deactivating the loadset, all new transactions would use version 1 of program ABCD. With this method, you can easily apply program updates and fixes while also having the ability to quickly and effectively revert to prior versions without a system outage or affecting the availability of business services.

7.5 System outage recovery

For a z/TPF system, a system outage means the system is no longer in a state where it is accepting or processing transactions and is IPLed (rebooted) as part of the recovery process. Regardless of whether a system outage was due to a malicious actor, an accident, or some other issue, the goal is to limit the duration and quickly get the system back to normal.

When an unplanned outage occurs, traffic is usually queued by other systems or the network outside of the z/TPF system. Those other systems typically hold on to that work, waiting to send it as soon as the z/TPF system is available to resume processing transactions. Even if some of that work times out and is discarded by those other systems, the amount of queued work could still be substantial.

When the z/TPF system resumes processing transactions, the queued work is typically released as a burst that can last for a few seconds to a few minutes. Depending on the amount of queued work, this burst could cause the z/TPF system to need more CPU resources than it normally needs or has and could cause some transactions to not be processed in a timely manner. This can result in the remote system timing out the transaction because the z/TPF did not respond quickly enough. In other words, even though the z/TPF system is back up and processing work again, an excessive surge in traffic that drives the system to 100% CPU busy can still result in business impacts.

For z/TPF systems running on an IBM z15 or later, you can use System Recovery Boost to provide extra CPU capacity, at no cost, starting in system IPL and for a period after z/TPF resumes processing transactions. The z/TPF system is designed to IPL quickly and efficiently within a few minutes, even without System Recovery Boost. As a result, the primary benefit of the extra CPU capacity is in helping work off that surge in traffic. By providing additional CPU resources during that initial surge of queued work, System Recovery Boost enables z/TPF to process all that work in a timely manner, preventing outage impacts from continuing even after the system is back up and accelerating the return to normal steady state processing.

7.6 Site failure recovery

Even if your data center is configured with multiple, redundant components and avoids any single point of failure, natural disasters, multiple failures, or malicious actors could result in being unable to run production workload at your primary data center.

IBM Redbook Accelerate Mainframe z/TPF Application Modernization with Hybrid Cloud, 5714-00 details three reference architectures for modernization. The first two reference architectures, the single-image z/TPF system and the active-active z/TPF cluster have a primary data center and use DASD replication to mirror the z/TPF database to a disaster recovery data center. To protect your business against natural disasters that could have a broad geographic impact, like hurricanes and earthquakes, the DR data center should be geographically distant from the primary data center.

If the primary data center fails and a disaster is declared, z/TPF systems are IPLed (rebooted) in the disaster recovery data center and network traffic is redirected to the disaster recovery site. If the z/TPF systems in the disaster recovery data center are running on an IBM z15 or later, they can take advantage of IBM System Recovery Boost to IPL and start processing transactions as quickly as possible.

The third reference architecture is the geographically dispersed active-active z/TPF systems, where multiple z/TPF systems are in two or more geographically distant data centers and use software replication to mirror z/TPF database updates among the z/TPF systems.

If one of the data centers failed and a disaster is declared, network traffic would be redirected to the remaining data centers.

In this scenario, each data center would be processing its normal workload plus a portion of the workload from the failed data center. As a result, advance planning is required to make sure additional capacity is available at each data center and ready to accommodate the extra workload in case of a disaster.

Regardless of which type of reference architecture you have adopted for your z/TPF system, a key aspect of managing site failures is having a documented disaster recovery plan and practicing that plan. By periodically practicing DR plans and following best practices, you provide a resilient solution for your business's most critical systems.



Conclusion

To improve your security posture, you should take the necessary steps to reduce risk and vulnerable exposure points. The z/TPF architecture has built-in security features and provides other mechanisms you can exploit at the application level and system level to protect your system and data, turning your z/TPF system into a fortress. IBM recommends the following best practices:

- ▶ Use pervasive encryption (encrypt all data in flight and at rest).
- ▶ Upgrade to an IBM z16 processor to use hardware accelerated strong cryptography to enable you to protect all data, even as your transaction volume increases.
- ▶ Use IBM provided middleware that is TLS enabled to communicate with external systems such that your applications can focus on the business logic and let the z/TPF system handle network security.
- ▶ Use automatic z/TPF database encryption such that your applications can focus on the business logic and let the z/TPF system handle database encryption.
- ▶ Create periodic crypto inventory reports to confirm the necessary network connections and databases are secure, and use cryptographic algorithms mandated by your internal security policy or regulatory compliance.
- ▶ Use centralized API management (APIM) to control and monitor access to all services in your enterprise, including those that are deployed on z/TPF.
- ▶ Protect your system as much as possible, but also have a plan for how to recover in the event of an attack using data vault technology such as IBM DS8000 IBM Safeguarded Copy.

Use this IBM Redbooks publication to review the different security procedures and capabilities that exist in the z/TPF environment and select the ones that drive the most immediate value for your business depending on your needs, environment, challenges, and current business processes.

By partnering with IBM and following our recommended security modernization journey, we can help make your environment more secure, make it difficult for a cyberattack to be successful, and protect your environment from many future attacks as well. The approaches and strategies that are presented in this publication can help you to accelerate your journey towards a truly secure environment.



A

z/TPF system and workload resiliency

This appendix provides information on system and workload resiliency. When your enterprise is running mission critical workload, the ability for your system and the transactions it is processing to withstand both expected and unexpected incidents is paramount. The topic of resiliency will be addressed from many different angles included those listed below that are all ingrained into the z/TPF architecture such that you get these benefits running your applications on z/TPF:

- ▶ Transaction isolation
- ▶ Memory protection
- ▶ Memory validation
- ▶ Program service restriction
- ▶ Database isolation
- ▶ High availability in a Java environment
- ▶ Workload spike protection
- ▶ High availability and hardware redundancy

For more information, see IBM Redbook *Accelerate Mainframe z/TPF Application Modernization with Hybrid Cloud*, 5714-00.

A.1 Transaction isolation

A common application architecture on other platforms is each application runs as a long-running multi-threaded process. Multiple transactions in these architectures share memory and therefore can see each other's data. In those cases, an application bug processing one transaction can corrupt data associated with a different transaction. In the worst-case scenario, it can cause the application process to fail. This impacts all transactions that are currently being processed as well as preventing the introduction of any new transactions until the application process can be restarted.

Figure A-1, shows a transaction flowing across many long-running multi-threaded processes within a server node. The Application 1 process might be processing dozens of transactions concurrently, where the data for all those transactions is visible to all the threads in Application Process 1. If processing a transaction results in an error that causes the process to fail, all the in-progress transactions are impacted. Until a new Application 1 process is started, new transactions are also impacted (cannot begin processing).

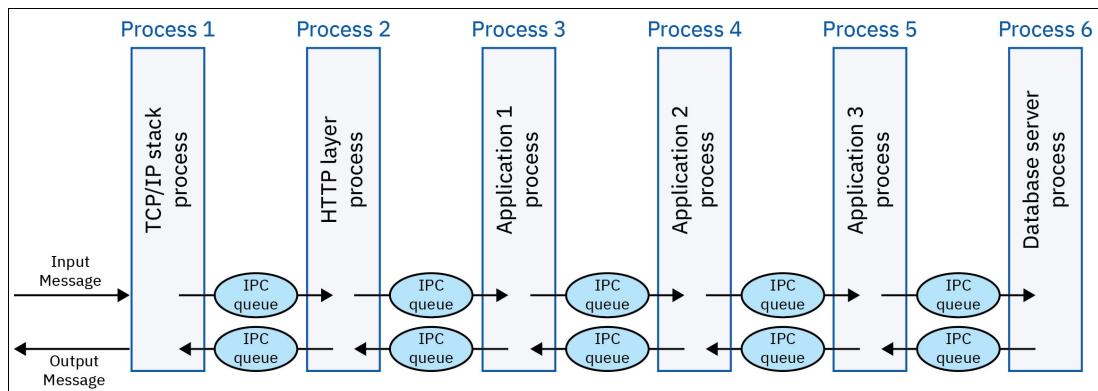


Figure A-1 Other platforms without transaction isolation

To avoid all those problems, z/TPF uses a different process execution model based on transaction isolation. One transaction cannot access or accidentally corrupt the data associated with another transaction currently being processed. If there is an error that causes the process associated with that transaction to terminate abnormally, only that one transaction is impacted instead of the entire application. As shown in Figure A-2 on page 91, each typical transaction on z/TPF runs in its own single-threaded process and all application and system components that are involved with processing that transaction share application memory. In the z/TPF model, true transaction isolation is achieved because each transaction runs in its own process. Since each transaction is isolated in its own process, it can see only its own data, not the data of any other transaction.

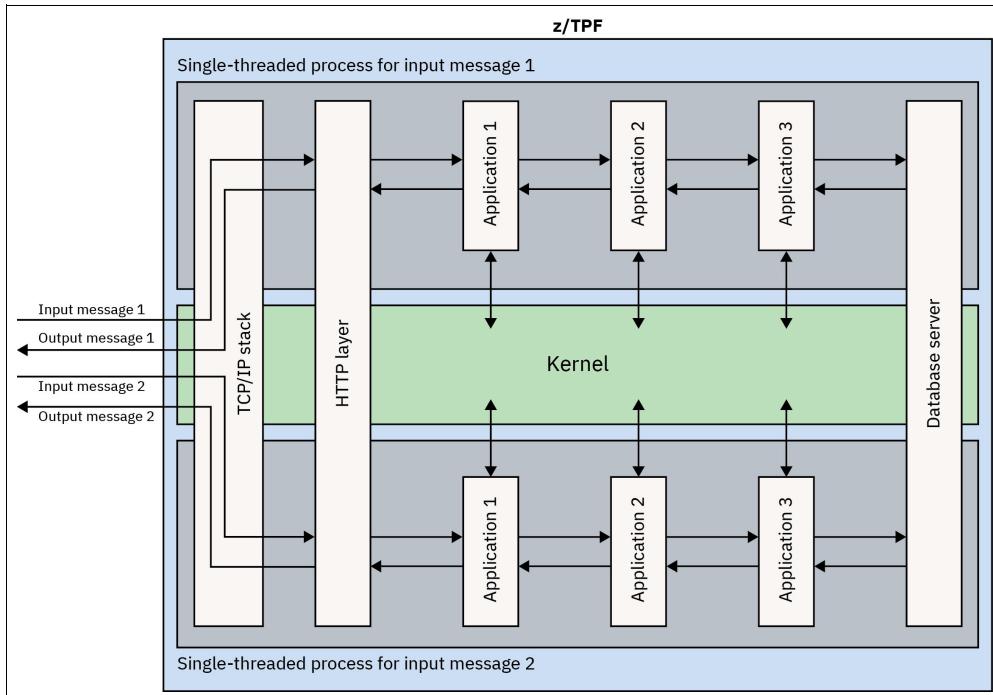


Figure A-2 Single-threaded process execution model on z/TPF

A.2 Memory protection

While the memory associated with a transaction is isolated from other transactions, there is certain data that might need to be shared by multiple transactions, and is accessed very frequently where for performance reasons the data needs to be in memory. Examples include:

- ▶ A memory table that contains all the 3-character airport codes and what city that represents (for example, airport code YYZ is Toronto, Canada)
- ▶ A memory table that contains application specific statistical information (for example, number of tickets sold in a given market)

Memory tables like this can be defined such that they are accessible by all transactions. Each area of memory on IBM Z is associated with a memory storage protect key. This key allows you to make some memory tables read-only to transactions to prevent accidental corruption of the data. Other tables can remain read-write if they are updated by applications. For example, Figure 5-7 on page 24 shows the memory view of 2 transactions, ECB 1 and ECB 2. Each ECB has transactional data that is only visible to that one ECB, all transactions (ECBs) can read and update shared user tables, but ECBs cannot update system tables. Bottom line, critical system tables use storage protect keys to prevent applications from accidentally corrupting those tables.

While processing a transaction, if the application program attempts to read memory that is not accessible to this process, or if the application program attempts to update memory for which this process has read-only access, z/TPF terminates the process because of a memory access violation. Only that one transaction fails while the application itself remains active and is able to process more transactions. The classic buffer overflow condition that enables an attacker to gain root access on some other platforms does exist on z/TPF. Instead, a buffer overflow would cause the z/TPF system to forcibly exit that one misbehaving application instance.

A.3 Memory validation

A more difficult application error to detect occurs when an application instance was using part of its memory, releases that memory, but a defect in the application accesses that memory afterwards. The results are unpredictable. If that memory has not been updated since it was released, the application logic might work. However, the memory might have been reused resulting in your application loading and then using the wrong data.

To catch these types of application errors during development and test on z/TPF, there are three memory validation modes (heap check mode, block check mode, and stack validation) that can be turned on to detect errors like an application program attempting to write beyond the end of a memory buffer area, or attempting to reference an area of memory after the program has released that memory. These should only be used on a z/TPF test system when unit testing your applications because they incur a lot of overhead that would negatively affect the performance of a production system.

A.4 Program service restriction

z/TPF provides many program services implemented as assembly macros and C function calls. These are divided into two categories, general purpose and restricted use. General purpose services can be called by application programs and the z/TPF systems code itself. By default, restricted use services can only be called by the z/TPF systems code itself because these services can damage the system if misused. If you have a business need for one of your application programs to use a restricted use service, the z/TPF operator can authorize that one program to be allowed to call restricted use services once proper justification is made.

A.5 Database isolation

A given database resides in a subsystem on z/TPF. In a multi-tenant environment such as multiple airlines being hosted on one z/TPF system, multiple database function (MDBF) on z/TPF provides database isolation preventing airline X from being able to access airline Y's data where each airline's databases reside in a different subsystem. For example, the databases for airline X reside in subsystem AIRX while the databases for airline Y reside in subsystem AIRY. When a transaction is being processed, the entry control block (ECB) associated with that application process is marked as running in a particular subsystem (like AIRX or AIRY). This allows one application to process transactions from users of different airlines. For example, an airline booking application service is coded to read and update the inventory record (INV) for a specified flight and then create a reservation record (PNR) for that passenger.

If the booking request is for airline X, the ECB is running in subsystem AIRX such that when it opens the INV record and creates the PNR, it will be using Airline X's version of those databases.

Figure A-3 shows two requests arriving to a z/TPF system with multiple subsystems defined. One request is a booking request for airline X so the ECB processing that transaction runs in the AIRX subsystem and the other is a booking request for airline Y so the ECB processing that transaction runs in the AIRY subsystem. While the two subsystems share the same booking applications, they are accessing and updating separate PNR databases.

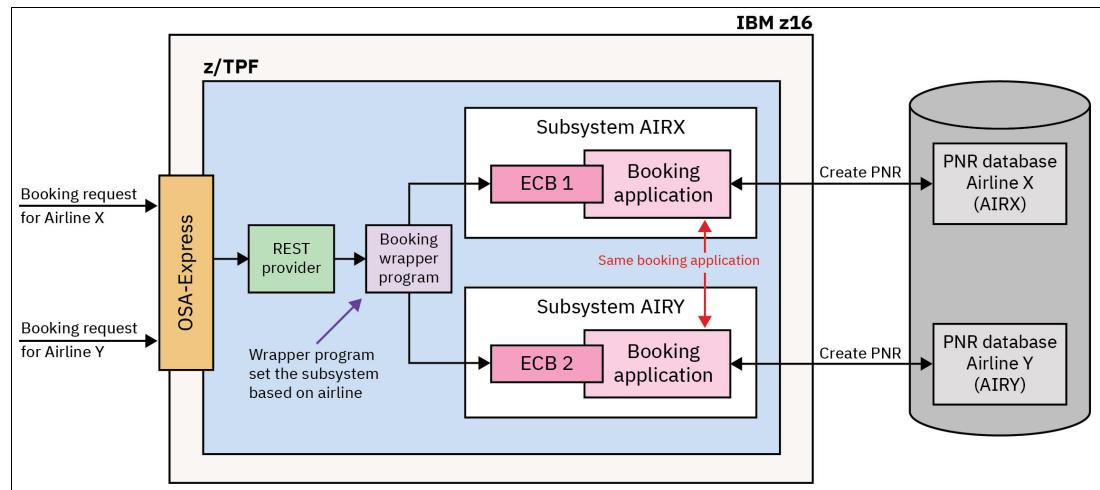


Figure A-3 Database isolation

A.6 High availability in a Java environment

Traditional z/TPF applications written in C/C++ or assembly language are typically designed where one process handles one transaction and then exits. However, applications written in Java execute in long-running processes (due to significant overhead required to start a JVM). Each process handles many transactions. This can be problematic because if one transaction causes a failure it can drive a full application outage.

To solve this, the z/TPF Application Manager for Java (JAM) facility provides high availability for Java applications on z/TPF where a JAM consists of multiple JVMs running the same Java application(s). The z/TPF system hides this complexity from users as the (local or remote) caller of a Java service on z/TPF only specifies the name of the service. The z/TPF system determines which JVM in the JAM to route that request to. If there is an error that causes one JVM to fail, the Java services are still available because there are other active JVMs that provide those services while the z/TPF system restarts the JVM that failed. The JAM concept also allows your workload to easily scale by defining additional JVMs to a JAM, like going from a JAM with 3 JVMs to a JAM that now has 4 JVMs.

For example, Figure A-4 shows a JAM with 3 JVMs running on a z/TPF system. This particular JAM is for the "Pricing" application and could contain several pricing related services, including the "GetPrice" service. When traditional z/TPF applications call the "GetPrice" service, the z/TPF system uses the "GetPrice" name to route the request to the "Pricing" JAM. From there, the z/TPF system automatically routes the request to the next available application thread in one of the JVMs. Because the next available thread processes the request, the request is processed in a timely fashion even if some of the other application threads or one of the JVMs is unavailable.

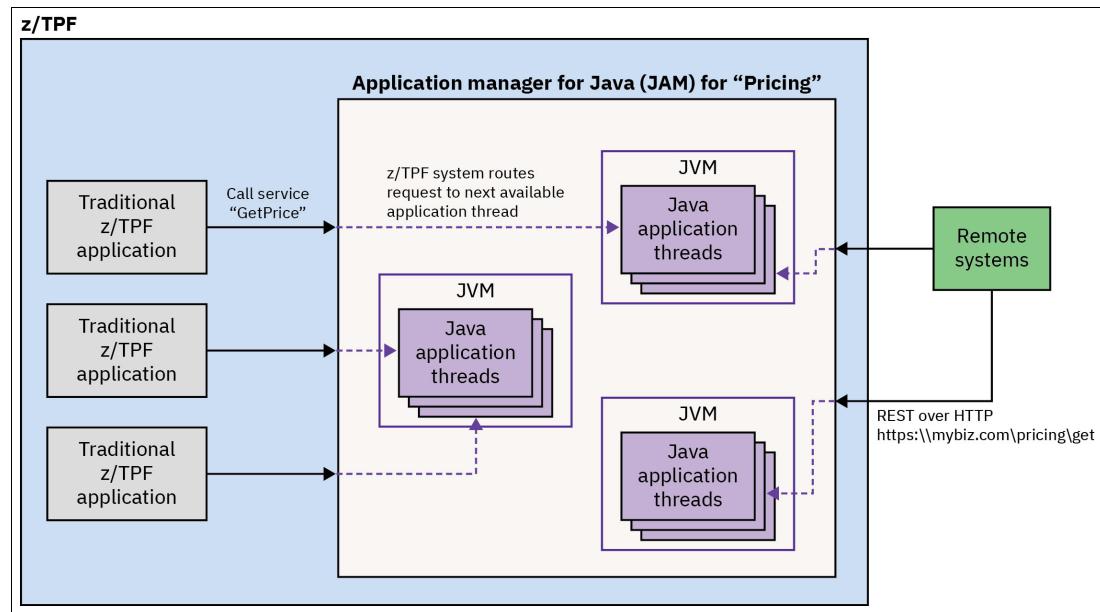


Figure A-4 Traditional z/TPF applications and remote systems calling a Java service in a JAM

Similar routing logic is done for remote callers as well, and Figure A-4 shows remote systems calling the same service using REST over HTTP. The z/TPF system routes the REST request to the same "Pricing" JAM, and just like the service calls from traditional z/TPF applications, the z/TPF system routes the request to the next available Java application thread.

For decades, the z/TPF dynamic program loader has enabled you to load a new version of a C/C++ or assembly application without any disruption or downtime. Transactions that were currently being processed continue to use the older version of the application and new transactions will use the new version of the application. If there are issues with the new version of the application, the z/TPF operator can quickly and easily fallback to the older version of the application. The JAM concept allows you to also dynamically load a new version of a Java application while maintaining that same 100% availability for the Java services managed by that JAM. The JVMs in a JAM are recycled one at a time where new requests are routed to an active JVM, allowing transactions in flight to finish processing while a new JVM using the new application code is created. Lastly, the old JVM is terminated.

A.7 Workload spike protection

Higher than normal transaction volumes on z/TPF can occur due to circumstances beyond your control. For example, if your z/TPF system handles airline reservations and there is bad weather in multiple hub cities, that can cause massive flight disruptions resulting in a spike in re-accommodation traffic for several hours.

If your z/TPF system handles credit card authorizations, a large retailer announcing a limited time sale can result in a surge of transactions on your z/TPF system that can also last several hours.

There are different methods you can use to ensure that your z/TPF production system can get sufficient CPU capacity to handle these workload spikes. Prioritizing workloads and the use of z/TPF Dynamic CPU are two methods to handle spikes.

The first method, workload prioritization, is to set up an environment where there are lower priority workloads (test systems, batch or other non-realtime traffic) running in LPARs in the same IBM Z processor as your z/TPF production system where these LPARs are configured to share physical CPU engines. If you assign the z/TPF production system LPAR the highest priority, you guarantee that system can get as much CPU capacity as it needs. The other lower priority LPARs (workloads) will then get to consume the remaining CPU capacity. If the surge in traffic on the production system is very high, or if your z/TPF production system is not sharing physical CPU engines with other workloads, this could be an indication that the current CPU capacity is not sufficient. In this case, you can use the second method, z/TPF Dynamic CPU support, to dynamically add additional CPU capacity to your production system without an outage.

A.8 HA and hardware redundancy

At a high-level, you want an architecture where there are no single points of failure. Components within a hardware device can fail (like a network adapter inside an IBM Z processor), and a hardware device (like a DASD control unit) can fail. A software bug can cause an application or the entire operating system to fail. A hardware or software failure could be accidental or the result of malicious act such as a cyberattack or disgruntled employee (insider threat). Regardless of the cause, you want an architecture that can survive the loss of any one component. Now more than ever, you also need to account for the case where Mother Nature renders a metropolitan area unusable for a period of weeks or even months.

IBM Z processors can make the loss of certain hardware components transparent to the operating systems (like z/TPF) running on that processor. For example, there are spare CPU cores to handle the case where a physical CPU core fails. Chips, cores, and memory are organized into drawers and a processor can have multiple drawers such that if one drawer fails, the remaining drawers are still able to run workloads. Memory has built in redundancy using Redundant Array of Independent Memory (RAIM) technology, and built in security as that memory is automatically encrypted on IBM Z.

IBM DASD also can handle certain hardware component failure transparently to the users of that DASD. The disks also can be configured to use Redundant Array of Independent Disk (RAID) technology to improve resiliency.

When creating your z/TPF system itself, there are several best practices to keep in mind when it comes to HA:

- ▶ Use multiple network (OSA-Express) adapters, each connected to a different physical Ethernet. Define virtual IP addresses (VIPAs) that can automatically be moved from one network adapter to another in case of an adapter failure.
 - Remote clients should connect using a z/TPF virtual IP address (VIPA) such that network failures are transparent to applications.
- ▶ Use duplicated data, meaning define 2 local copies of all your data (prime and duplicate copies) that reside on different DASD control units.

- ▶ Use multiple tape grids.
- ▶ Have multiple operator consoles defined, and multiple fallback consoles for primary operator console (prime CRAS) in the data center.
- ▶ Use multiple Crypto-Express adapters for RSA public key cryptography operations.
- ▶ Have multiple z/TPF systems and either:
 - Create an Active-Active z/TPF environment, meaning a loosely coupled cluster of multiple z/TPF systems, all processing transactions concurrently in the same data center sharing the same database. Have enough CPU capacity where if you were to lose one or two z/TPF systems, the remaining z/TPF systems can still handle the entire workload. Have one or more D/R sites that are in different geographies (hundreds to thousands of miles away from the primary data center) and use DASD replication services to keep a near current copy of your data in each DR site.
 - Create a Geographically Dispersed Active-Active z/TPF environments, meaning multiple z/TPF systems, each with their own copy of the database, all processing transactions concurrently in data centers in different geographies. Have enough CPU capacity where if you were to lose a data center, the z/TPF systems in the remaining data centers are able to process the entire workload. You would have to implement host level database replication to keep the copies of the databases in sync, and because this is an eventually consistent database architecture, you have to write code to handle and resolve data conflicts (applications in two or more z/TPF systems update the same database record at roughly the same time, then replicate those changes to the other systems).

Identify your threat targets

This appendix provides information about the importance of identifying threat targets. Whichever threat modeling tools you use, it is important to understand what these threat targets are so you can better protect yourself. Let us take a look at a typical request/reply transaction flow through an enterprise to better understand what common threat targets are. Figure B-1 shows the transaction flow from an end user, interfacing with a public cloud that then interfaces with the backend application servers in your data center to access and update the system of record data.

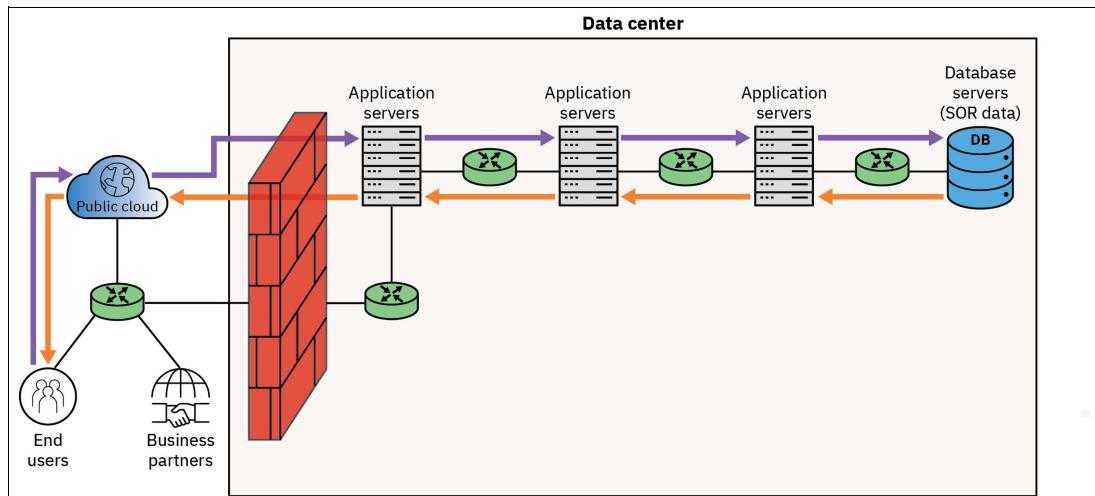


Figure B-1 Sample transaction request/reply flow

When a transaction completes, some of the database updates may be replicated to other database servers that serve as a read-only copy of this data for use by other parts of your business. Figure B-2 on page 98 shows the post-transaction propagation of data to read-only database servers. This example shows the database servers being on premises, but some could be remote such as in a public cloud.

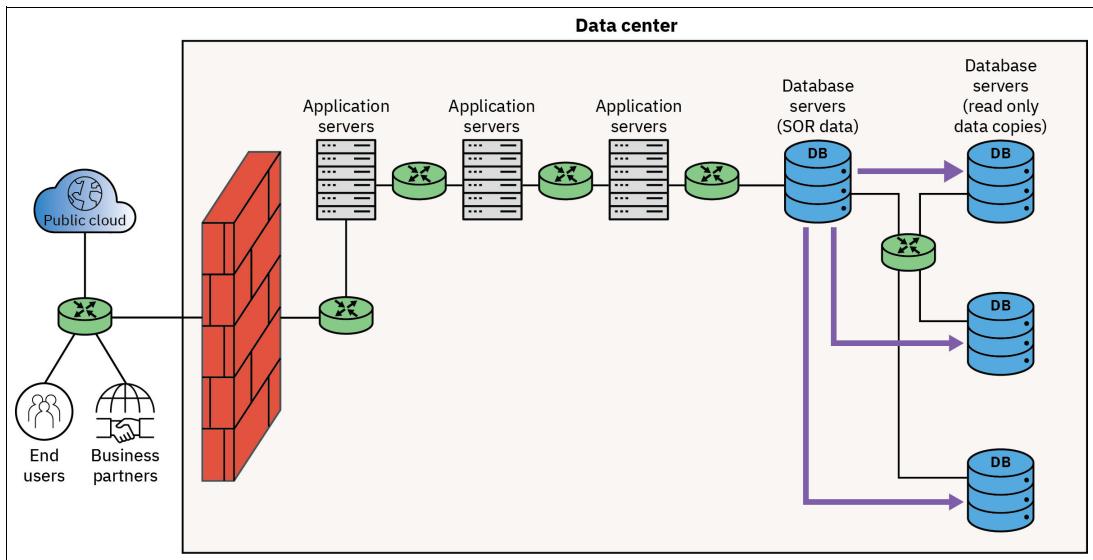


Figure B-2 Post transaction - some database updates propagated to other databases

As described, other parts of your business may interface with the read-only copies of data. These are typically used for high volume query-only type transactions, or database intensive analytics to optimize your business that runs on separate systems to avoid impacting the real-time revenue generating transaction workloads. Figure B-3 shows other employees and application servers working against the read-only copies of your data.

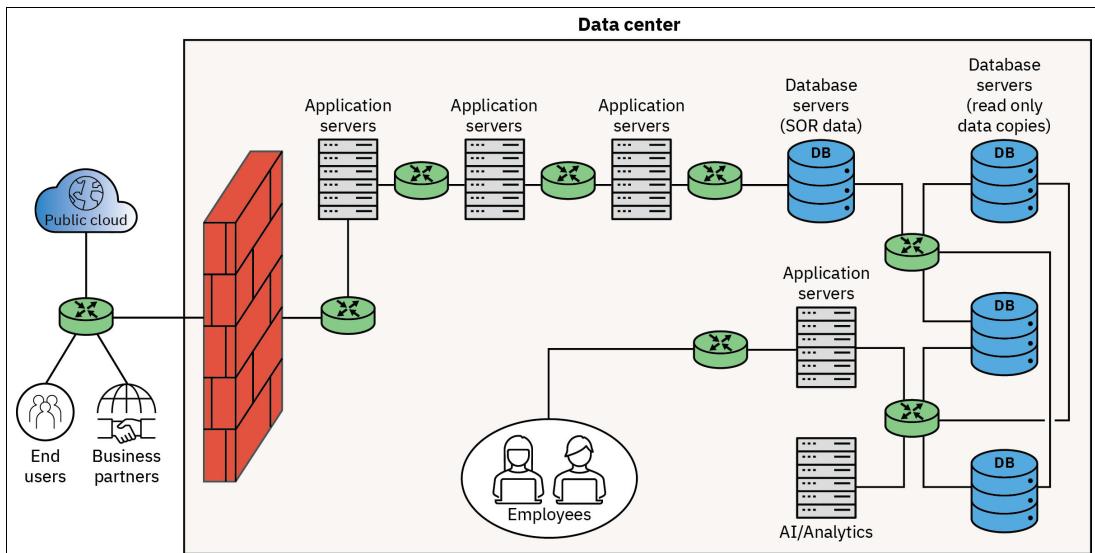


Figure B-3 Other parts of your business work against read-only copies of data

In an environment like this, the first threat target to be aware of is the multiple network routers throughout your enterprise. Figure B-4 on page 99 shows the numerous network routers inside your enterprise (data center) as well as external routers that could be vulnerable to an attack. The types of threats that can be exploited in your network routers may consist of the following:

- ▶ Viewing data in unsecure network traffic.
- ▶ Recording all your traffic for exposure to the data at a future time (harvest now, decrypt later).

- ▶ Man-in-the-Middle attacks where you think you are communicating with a server but you are talking to a malicious code acting as the server.
- ▶ Re-routing of traffic to the hijacker's server.

All these types of attacks can take place within your network infrastructure.

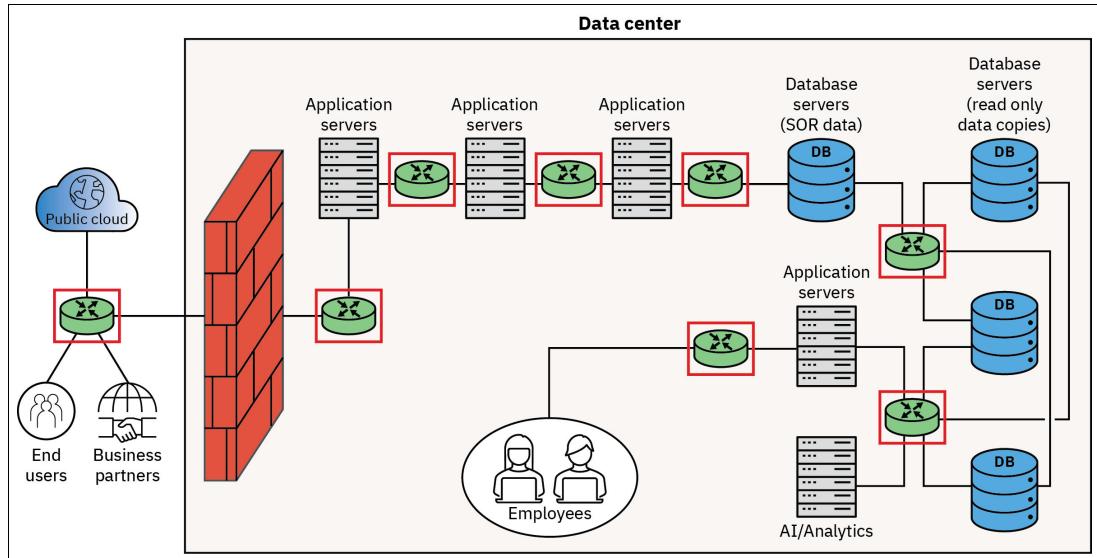


Figure B-4 Threat target - network routers

Figure B-5 highlights the application servers within your enterprise and in the external public cloud that can be vulnerable to threats. Every transaction passes through multiple application servers where attackers might try to inject malicious code to corrupt or steal your data, or impact the server itself to deny service to other legitimate users. The malicious code could also perform passive attacks where all transactions and data that flows through these application servers is recorded and made available to the attacker. The application servers that work with the read-only copies of the data cannot corrupt that data. However, malicious code in those servers could steal your data or alter the processing of that data to produce false reports or cause the wrong actions to be taken.

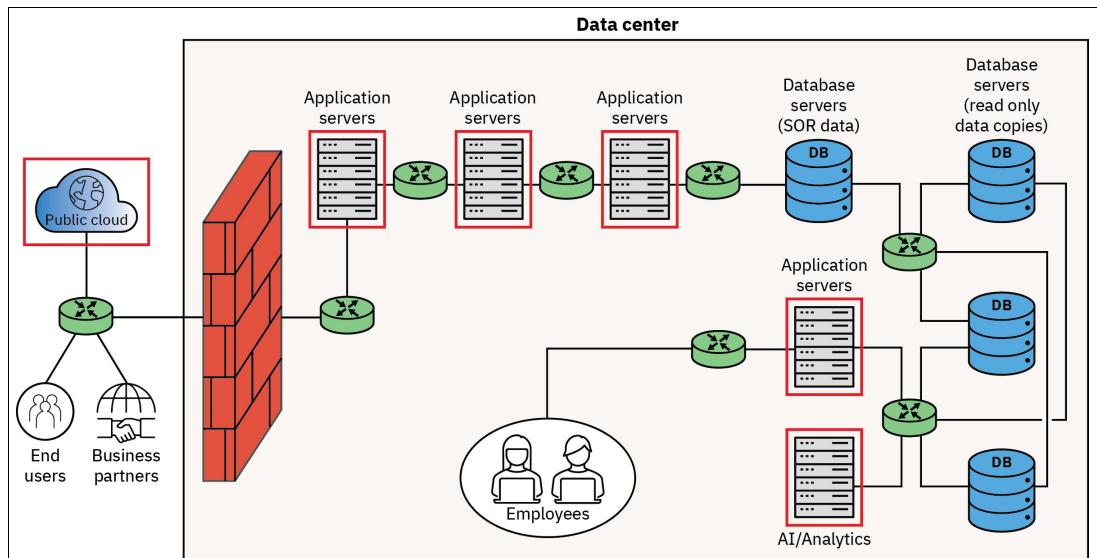


Figure B-5 Threat target - application servers

Figure B-6 highlights the database servers within your enterprise. The database servers store all your data, including personal and sensitive data. Threats against your database servers include stealing all or some of the sensitive data retained on these servers, corrupting the data to deny service to other users, or in some cases hold your entire database and in turn your business hostage by using a ransomware attack.

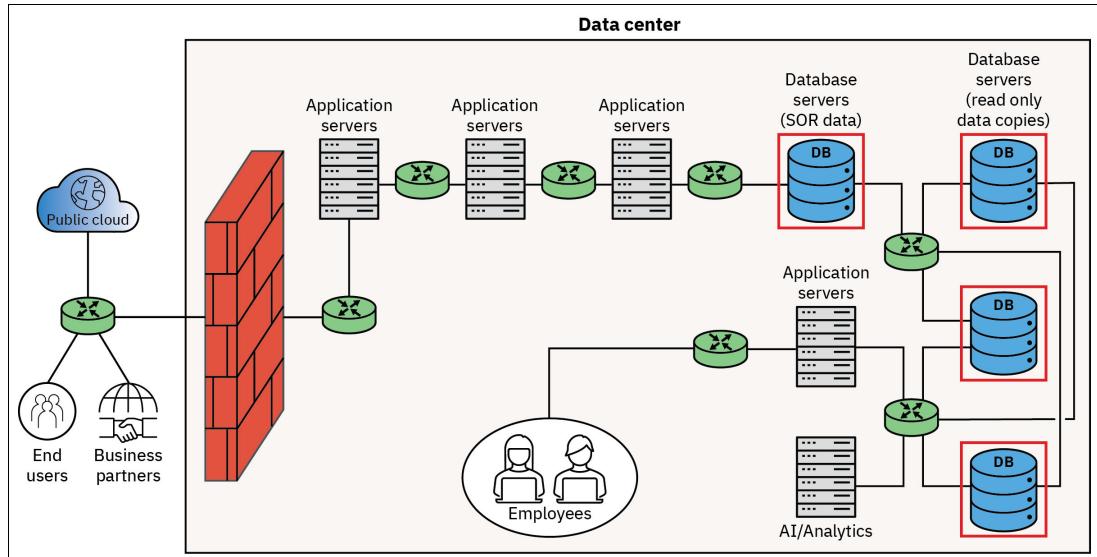


Figure B-6 Threat target - database servers

Figure B-7 highlights the human vulnerability, your own employees. You must protect against inside threat attacks where a rogue employee can harm your business, and from bad actors that are impersonating an employee to gain access to your systems. Threats from inside your business include hijacking "other" employees' accounts, granting access to outside hijackers, and infecting systems to access parts of your business and mount future attacks.

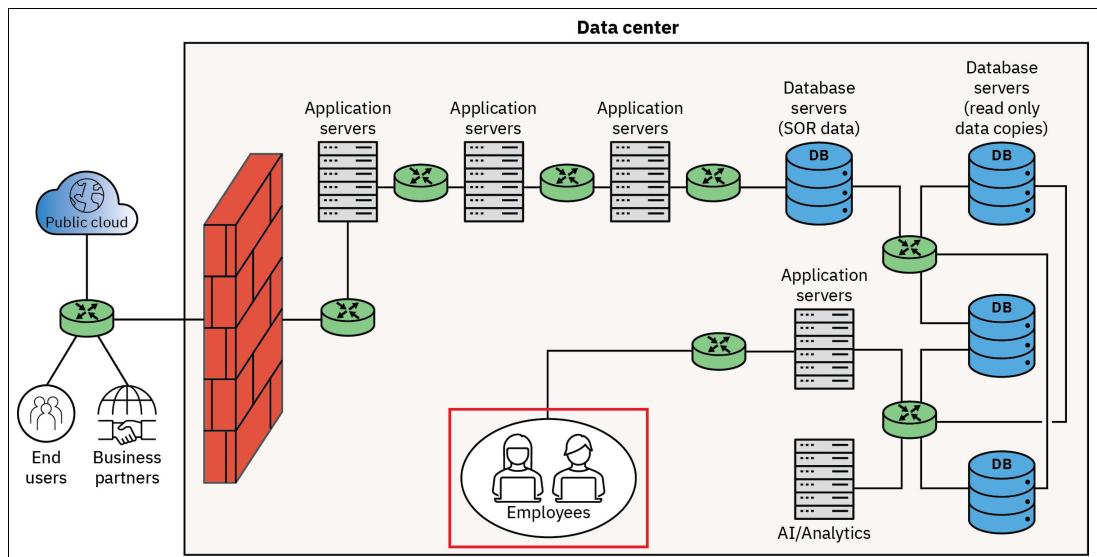


Figure B-7 Threat target - your employees

Even your business partners can be a threat target to your business. Figure B-8 highlights the end users and business partners that can be a threat. Bad actors within your end users and business partners as well as attackers that steal end user credentials can impersonate clients to access or steal your data. Another threat is driving fraudulent transactions to impact system performance, deny service for other users (DoS attacks), and other business impacts.

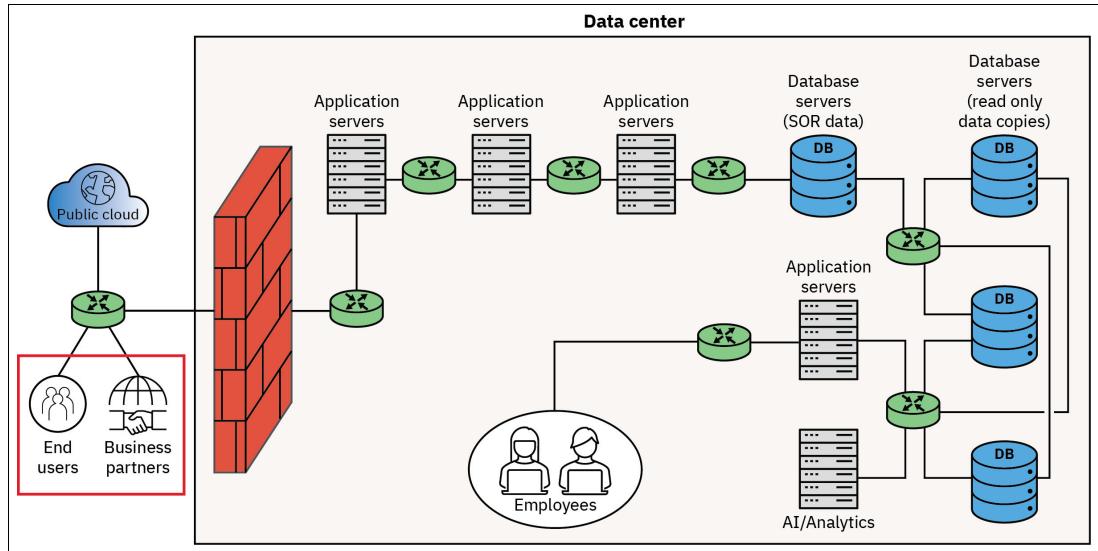


Figure B-8 Threat target - your clients and partners



REDP-5740-00

ISBN073846158X07

Printed in U.S.A.

Get connected

