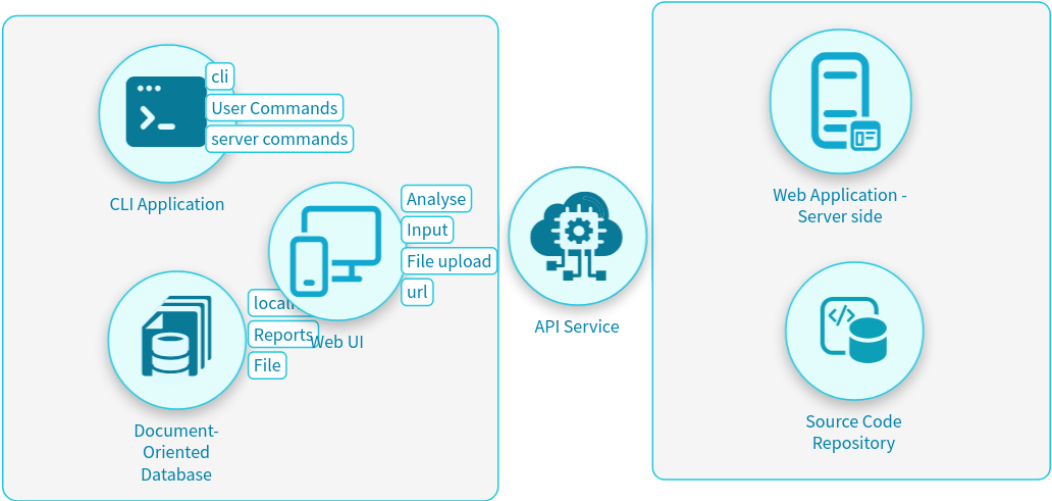


CodeSentinel Thread Modeling

PCI-DSS-v3.2.1 - Compliance report

Fri Mar 14 2025 18:54:57 GMT+0000 (Coordinated Universal Time)

Project description: No description
Unique ID: codesentinel-thread-modeling-1741975253263
Owner: [Taha Juzar]
Workflow state: Draft
Tags: No tags



Compliance summary

See the countermeasure state data categorized by standards:

PCI-DSS-v3.2.1: 1.1.1			PCI-DSS-v3.2.1: 1.1.2			PCI-DSS-v3.2.1: 1.1.3		
Implemented	<div><div></div></div>	0	Implemented	<div><div></div></div>	0	Implemented	<div><div></div></div>	0
Required	<div><div></div></div>	0	Required	<div><div></div></div>	0	Required	<div><div></div></div>	0
Failed	<div><div></div></div>	0	Failed	<div><div></div></div>	0	Failed	<div><div></div></div>	0
Verified	<div><div></div></div>	0	Verified	<div><div></div></div>	0	Verified	<div><div></div></div>	0
Not-applicable	<div><div></div></div>	0	Not-applicable	<div><div></div></div>	0	Not-applicable	<div><div></div></div>	0
Recommended	<div><div></div></div>	3	Recommended	<div><div></div></div>	3	Recommended	<div><div></div></div>	3
Rejected	<div><div></div></div>	0	Rejected	<div><div></div></div>	0	Rejected	<div><div></div></div>	0
Non-compliant	<div><div></div></div>	3	Non-compliant	<div><div></div></div>	3	Non-compliant	<div><div></div></div>	3

PCI-DSS-v3.2.1: 10.6.1			PCI-DSS-v3.2.1: 10.6.2			PCI-DSS-v3.2.1: 10.6.3		
Implemented	<div><div></div></div>	0	Implemented	<div><div></div></div>	0	Implemented	<div><div></div></div>	0
Required	<div><div></div></div>	0	Required	<div><div></div></div>	0	Required	<div><div></div></div>	0
Failed	<div><div></div></div>	0	Failed	<div><div></div></div>	0	Failed	<div><div></div></div>	0
Verified	<div><div></div></div>	0	Verified	<div><div></div></div>	0	Verified	<div><div></div></div>	0
Not-applicable	<div><div></div></div>	0	Not-applicable	<div><div></div></div>	0	Not-applicable	<div><div></div></div>	0
Recommended	<div><div></div></div>	6	Recommended	<div><div></div></div>	6	Recommended	<div><div></div></div>	3
Rejected	<div><div></div></div>	0	Rejected	<div><div></div></div>	0	Rejected	<div><div></div></div>	0
Non-compliant	<div><div></div></div>	6	Non-compliant	<div><div></div></div>	6	Non-compliant	<div><div></div></div>	3

PCI-DSS-v3.2.1: 11.1.1			PCI-DSS-v3.2.1: 11.5.1			PCI-DSS-v3.2.1: 12.10.5		
Implemented	<div><div></div></div>	0	Implemented	<div><div></div></div>	0	Implemented	<div><div></div></div>	0
Required	<div><div></div></div>	0	Required	<div><div></div></div>	0	Required	<div><div></div></div>	0
Failed	<div><div></div></div>	0	Failed	<div><div></div></div>	0	Failed	<div><div></div></div>	0
Verified	<div><div></div></div>	0	Verified	<div><div></div></div>	0	Verified	<div><div></div></div>	0
Not-applicable	<div><div></div></div>	0	Not-applicable	<div><div></div></div>	0	Not-applicable	<div><div></div></div>	0
Recommended	<div><div></div></div>	2	Recommended	<div><div></div></div>	3	Recommended	<div><div></div></div>	5
Rejected	<div><div></div></div>	0	Rejected	<div><div></div></div>	0	Rejected	<div><div></div></div>	0
Non-compliant	<div><div></div></div>	2	Non-compliant	<div><div></div></div>	3	Non-compliant	<div><div></div></div>	5

PCI-DSS-v3.2.1: 12.3.10			PCI-DSS-v3.2.1: 12.3.3			PCI-DSS-v3.2.1: 12.3.7		
Implemented	<div><div></div></div>	0	Implemented	<div><div></div></div>	0	Implemented	<div><div></div></div>	0
Required	<div><div></div></div>	0	Required	<div><div></div></div>	0	Required	<div><div></div></div>	0
Failed	<div><div></div></div>	0	Failed	<div><div></div></div>	0	Failed	<div><div></div></div>	0
Verified	<div><div></div></div>	0	Verified	<div><div></div></div>	0	Verified	<div><div></div></div>	0
Not-applicable	<div><div></div></div>	0	Not-applicable	<div><div></div></div>	0	Not-applicable	<div><div></div></div>	0
Recommended	<div><div></div></div>	3	Recommended	<div><div></div></div>	2	Recommended	<div><div></div></div>	2
Rejected	<div><div></div></div>	0	Rejected	<div><div></div></div>	0	Rejected	<div><div></div></div>	0
Non-compliant	<div><div></div></div>	3	Non-compliant	<div><div></div></div>	2	Non-compliant	<div><div></div></div>	2

PCI-DSS-v3.2.1: 12.3.8			PCI-DSS-v3.2.1: 12.3.9			PCI-DSS-v3.2.1: 12.5.2		
✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0
🟡 Required	<div></div>	0	🟡 Required	<div></div>	0	🟡 Required	<div></div>	0
❌ Failed	<div></div>	0	❌ Failed	<div></div>	0	❌ Failed	<div></div>	0
✅ Verified	<div></div>	0	✅ Verified	<div></div>	0	✅ Verified	<div></div>	0
⚪ Not-applicable	<div></div>	0	⚪ Not-applicable	<div></div>	0	⚪ Not-applicable	<div></div>	0
🔵 Recommended	<div></div>	3	🔵 Recommended	<div></div>	3	🔵 Recommended	<div></div>	3
❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0
⚪ Non-compliant	<div></div>	3	⚪ Non-compliant	<div></div>	3	⚪ Non-compliant	<div></div>	3
PCI-DSS-v3.2.1: 6.4.1			PCI-DSS-v3.2.1: 6.4.2			PCI-DSS-v3.2.1: 7.1.4		
✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0
🟡 Required	<div></div>	0	🟡 Required	<div></div>	0	🟡 Required	<div></div>	0
❌ Failed	<div></div>	0	❌ Failed	<div></div>	0	❌ Failed	<div></div>	0
✅ Verified	<div></div>	0	✅ Verified	<div></div>	0	✅ Verified	<div></div>	0
⚪ Not-applicable	<div></div>	0	⚪ Not-applicable	<div></div>	0	⚪ Not-applicable	<div></div>	0
🔵 Recommended	<div></div>	1	🔵 Recommended	<div></div>	4	🔵 Recommended	<div></div>	2
❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0
⚪ Non-compliant	<div></div>	1	⚪ Non-compliant	<div></div>	4	⚪ Non-compliant	<div></div>	2
PCI-DSS-v3.2.1: 8.1.5			PCI-DSS-v3.2.1: 8.2.1			PCI-DSS-v3.2.1: 8.2.2		
✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0
🟡 Required	<div></div>	0	🟡 Required	<div></div>	0	🟡 Required	<div></div>	0
❌ Failed	<div></div>	0	❌ Failed	<div></div>	0	❌ Failed	<div></div>	0
✅ Verified	<div></div>	0	✅ Verified	<div></div>	0	✅ Verified	<div></div>	0
⚪ Not-applicable	<div></div>	0	⚪ Not-applicable	<div></div>	0	⚪ Not-applicable	<div></div>	0
🔵 Recommended	<div></div>	3	🔵 Recommended	<div></div>	3	🔵 Recommended	<div></div>	2
❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0
⚪ Non-compliant	<div></div>	3	⚪ Non-compliant	<div></div>	3	⚪ Non-compliant	<div></div>	2
PCI-DSS-v3.2.1: 8.5.1			PCI-DSS-v3.2.1: 9.1.1			PCI-DSS-v3.2.1: 1		
✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0
🟡 Required	<div></div>	0	🟡 Required	<div></div>	0	🟡 Required	<div></div>	0
❌ Failed	<div></div>	0	❌ Failed	<div></div>	0	❌ Failed	<div></div>	0
✅ Verified	<div></div>	0	✅ Verified	<div></div>	0	✅ Verified	<div></div>	0
⚪ Not-applicable	<div></div>	0	⚪ Not-applicable	<div></div>	0	⚪ Not-applicable	<div></div>	0
🔵 Recommended	<div></div>	3	🔵 Recommended	<div></div>	6	🔵 Recommended	<div></div>	2
❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0
⚪ Non-compliant	<div></div>	3	⚪ Non-compliant	<div></div>	6	⚪ Non-compliant	<div></div>	2
PCI-DSS-v3.2.1: 1.1			PCI-DSS-v3.2.1: 1.2			PCI-DSS-v3.2.1: 1.3		
✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0
🟡 Required	<div></div>	0	🟡 Required	<div></div>	0	🟡 Required	<div></div>	0
❌ Failed	<div></div>	0	❌ Failed	<div></div>	0	❌ Failed	<div></div>	0
✅ Verified	<div></div>	0	✅ Verified	<div></div>	0	✅ Verified	<div></div>	0
⚪ Not-applicable	<div></div>	0	⚪ Not-applicable	<div></div>	0	⚪ Not-applicable	<div></div>	0
🔵 Recommended	<div></div>	2	🔵 Recommended	<div></div>	5	🔵 Recommended	<div></div>	2
❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0
⚪ Non-compliant	<div></div>	2	⚪ Non-compliant	<div></div>	5	⚪ Non-compliant	<div></div>	2

PCI-DSS-v3.2.1: 1.5			PCI-DSS-v3.2.1: 2			PCI-DSS-v3.2.1: 2.1		
✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0
🟡 Required	<div></div>	0	🟡 Required	<div></div>	0	🟡 Required	<div></div>	0
❌ Failed	<div></div>	0	❌ Failed	<div></div>	0	❌ Failed	<div></div>	0
🔍 Verified	<div></div>	0	🔍 Verified	<div></div>	0	🔍 Verified	<div></div>	0
⚠ Not-applicable	<div></div>	0	⚠ Not-applicable	<div></div>	0	⚠ Not-applicable	<div></div>	0
🔵 Recommended	<div></div>	1	🔵 Recommended	<div></div>	2	🔵 Recommended	<div></div>	2
❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0
🔴 Non-compliant	<div></div>	1	🔴 Non-compliant	<div></div>	2	🔴 Non-compliant	<div></div>	2
PCI-DSS-v3.2.1: 2.2			PCI-DSS-v3.2.1: 2.3			PCI-DSS-v3.2.1: 2.4		
✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0
🟡 Required	<div></div>	0	🟡 Required	<div></div>	0	🟡 Required	<div></div>	0
❌ Failed	<div></div>	0	❌ Failed	<div></div>	0	❌ Failed	<div></div>	0
🔍 Verified	<div></div>	0	🔍 Verified	<div></div>	0	🔍 Verified	<div></div>	0
⚠ Not-applicable	<div></div>	0	⚠ Not-applicable	<div></div>	0	⚠ Not-applicable	<div></div>	0
🔵 Recommended	<div></div>	6	🔵 Recommended	<div></div>	3	🔵 Recommended	<div></div>	3
❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0
🔴 Non-compliant	<div></div>	6	🔴 Non-compliant	<div></div>	3	🔴 Non-compliant	<div></div>	3
PCI-DSS-v3.2.1: 2.5			PCI-DSS-v3.2.1: 3			PCI-DSS-v3.2.1: 3.7		
✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0
🟡 Required	<div></div>	0	🟡 Required	<div></div>	0	🟡 Required	<div></div>	0
❌ Failed	<div></div>	0	❌ Failed	<div></div>	0	❌ Failed	<div></div>	0
🔍 Verified	<div></div>	0	🔍 Verified	<div></div>	0	🔍 Verified	<div></div>	0
⚠ Not-applicable	<div></div>	0	⚠ Not-applicable	<div></div>	0	⚠ Not-applicable	<div></div>	0
🔵 Recommended	<div></div>	1	🔵 Recommended	<div></div>	2	🔵 Recommended	<div></div>	1
❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0
🔴 Non-compliant	<div></div>	1	🔴 Non-compliant	<div></div>	2	🔴 Non-compliant	<div></div>	1
PCI-DSS-v3.2.1: 4			PCI-DSS-v3.2.1: 4.3			PCI-DSS-v3.2.1: 5.4		
✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0
🟡 Required	<div></div>	0	🟡 Required	<div></div>	0	🟡 Required	<div></div>	0
❌ Failed	<div></div>	0	❌ Failed	<div></div>	0	❌ Failed	<div></div>	0
🔍 Verified	<div></div>	0	🔍 Verified	<div></div>	0	🔍 Verified	<div></div>	0
⚠ Not-applicable	<div></div>	0	⚠ Not-applicable	<div></div>	0	⚠ Not-applicable	<div></div>	0
🔵 Recommended	<div></div>	1	🔵 Recommended	<div></div>	1	🔵 Recommended	<div></div>	1
❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0
🔴 Non-compliant	<div></div>	1	🔴 Non-compliant	<div></div>	1	🔴 Non-compliant	<div></div>	1
PCI-DSS-v3.2.1: 6.1			PCI-DSS-v3.2.1: 6.2			PCI-DSS-v3.2.1: 6.4		
✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0
🟡 Required	<div></div>	0	🟡 Required	<div></div>	0	🟡 Required	<div></div>	0
❌ Failed	<div></div>	0	❌ Failed	<div></div>	0	❌ Failed	<div></div>	0
🔍 Verified	<div></div>	0	🔍 Verified	<div></div>	0	🔍 Verified	<div></div>	0
⚠ Not-applicable	<div></div>	0	⚠ Not-applicable	<div></div>	0	⚠ Not-applicable	<div></div>	0
🔵 Recommended	<div></div>	1	🔵 Recommended	<div></div>	3	🔵 Recommended	<div></div>	2
❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0
🔴 Non-compliant	<div></div>	1	🔴 Non-compliant	<div></div>	3	🔴 Non-compliant	<div></div>	2

PCI-DSS-v3.2.1: 6.5			PCI-DSS-v3.2.1: 6.7			PCI-DSS-v3.2.1: 7.1		
✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0
⚠ Required	<div></div>	0	⚠ Required	<div></div>	0	⚠ Required	<div></div>	0
❌ Failed	<div></div>	0	❌ Failed	<div></div>	0	❌ Failed	<div></div>	0
✅ Verified	<div></div>	0	✅ Verified	<div></div>	0	✅ Verified	<div></div>	0
⚡ Not-applicable	<div></div>	0	⚡ Not-applicable	<div></div>	0	⚡ Not-applicable	<div></div>	0
🔒 Recommended	<div></div>	1	🔒 Recommended	<div></div>	1	🔒 Recommended	<div></div>	5
❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0
⚠ Non-compliant	<div></div>	1	⚠ Non-compliant	<div></div>	1	⚠ Non-compliant	<div></div>	5
PCI-DSS-v3.2.1: 7.2			PCI-DSS-v3.2.1: 7.3			PCI-DSS-v3.2.1: 8.1		
✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0
⚠ Required	<div></div>	0	⚠ Required	<div></div>	0	⚠ Required	<div></div>	0
❌ Failed	<div></div>	0	❌ Failed	<div></div>	0	❌ Failed	<div></div>	0
✅ Verified	<div></div>	0	✅ Verified	<div></div>	0	✅ Verified	<div></div>	0
⚡ Not-applicable	<div></div>	0	⚡ Not-applicable	<div></div>	0	⚡ Not-applicable	<div></div>	0
🔒 Recommended	<div></div>	5	🔒 Recommended	<div></div>	1	🔒 Recommended	<div></div>	3
❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0
⚠ Non-compliant	<div></div>	5	⚠ Non-compliant	<div></div>	1	⚠ Non-compliant	<div></div>	3
PCI-DSS-v3.2.1: 8.2			PCI-DSS-v3.2.1: 8.3			PCI-DSS-v3.2.1: 8.5		
✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0
⚠ Required	<div></div>	0	⚠ Required	<div></div>	0	⚠ Required	<div></div>	0
❌ Failed	<div></div>	0	❌ Failed	<div></div>	0	❌ Failed	<div></div>	0
✅ Verified	<div></div>	0	✅ Verified	<div></div>	0	✅ Verified	<div></div>	0
⚡ Not-applicable	<div></div>	0	⚡ Not-applicable	<div></div>	0	⚡ Not-applicable	<div></div>	0
🔒 Recommended	<div></div>	3	🔒 Recommended	<div></div>	5	🔒 Recommended	<div></div>	2
❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0
⚠ Non-compliant	<div></div>	3	⚠ Non-compliant	<div></div>	5	⚠ Non-compliant	<div></div>	2
PCI-DSS-v3.2.1: 8.6			PCI-DSS-v3.2.1: 8.7			PCI-DSS-v3.2.1: 8.8		
✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0
⚠ Required	<div></div>	0	⚠ Required	<div></div>	0	⚠ Required	<div></div>	0
❌ Failed	<div></div>	0	❌ Failed	<div></div>	0	❌ Failed	<div></div>	0
✅ Verified	<div></div>	0	✅ Verified	<div></div>	0	✅ Verified	<div></div>	0
⚡ Not-applicable	<div></div>	0	⚡ Not-applicable	<div></div>	0	⚡ Not-applicable	<div></div>	0
🔒 Recommended	<div></div>	2	🔒 Recommended	<div></div>	3	🔒 Recommended	<div></div>	1
❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0
⚠ Non-compliant	<div></div>	2	⚠ Non-compliant	<div></div>	3	⚠ Non-compliant	<div></div>	1
PCI-DSS-v3.2.1: 9.10			PCI-DSS-v3.2.1: 9.3			PCI-DSS-v3.2.1: 9.5		
✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0
⚠ Required	<div></div>	0	⚠ Required	<div></div>	0	⚠ Required	<div></div>	0
❌ Failed	<div></div>	0	❌ Failed	<div></div>	0	❌ Failed	<div></div>	0
✅ Verified	<div></div>	0	✅ Verified	<div></div>	0	✅ Verified	<div></div>	0
⚡ Not-applicable	<div></div>	0	⚡ Not-applicable	<div></div>	0	⚡ Not-applicable	<div></div>	0
🔒 Recommended	<div></div>	1	🔒 Recommended	<div></div>	5	🔒 Recommended	<div></div>	2
❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0
⚠ Non-compliant	<div></div>	1	⚠ Non-compliant	<div></div>	5	⚠ Non-compliant	<div></div>	2

PCI-DSS-v3.2.1: 9.6			PCI-DSS-v3.2.1: 9.7			PCI-DSS-v3.2.1: 9.8		
✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0
🟡 Required	<div></div>	0	🟡 Required	<div></div>	0	🟡 Required	<div></div>	0
❌ Failed	<div></div>	0	❌ Failed	<div></div>	0	❌ Failed	<div></div>	0
🔍 Verified	<div></div>	0	🔍 Verified	<div></div>	0	🔍 Verified	<div></div>	0
⚪ Not-applicable	<div></div>	0	⚪ Not-applicable	<div></div>	0	⚪ Not-applicable	<div></div>	0
🟢 Recommended	<div></div>	2	🟢 Recommended	<div></div>	2	🟢 Recommended	<div></div>	2
❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0
⚪ Non-compliant	<div></div>	2	⚪ Non-compliant	<div></div>	2	⚪ Non-compliant	<div></div>	2
PCI-DSS-v3.2.1: 9.9			PCI-DSS-v3.2.1: 10.1			PCI-DSS-v3.2.1: 10.2		
✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0
🟡 Required	<div></div>	0	🟡 Required	<div></div>	0	🟡 Required	<div></div>	0
❌ Failed	<div></div>	0	❌ Failed	<div></div>	0	❌ Failed	<div></div>	0
🔍 Verified	<div></div>	0	🔍 Verified	<div></div>	0	🔍 Verified	<div></div>	0
⚪ Not-applicable	<div></div>	0	⚪ Not-applicable	<div></div>	0	⚪ Not-applicable	<div></div>	0
🟢 Recommended	<div></div>	2	🟢 Recommended	<div></div>	5	🟢 Recommended	<div></div>	3
❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0
⚪ Non-compliant	<div></div>	2	⚪ Non-compliant	<div></div>	5	⚪ Non-compliant	<div></div>	3
PCI-DSS-v3.2.1: 10.3			PCI-DSS-v3.2.1: 10.4			PCI-DSS-v3.2.1: 10.5		
✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0
🟡 Required	<div></div>	0	🟡 Required	<div></div>	0	🟡 Required	<div></div>	0
❌ Failed	<div></div>	0	❌ Failed	<div></div>	0	❌ Failed	<div></div>	0
🔍 Verified	<div></div>	0	🔍 Verified	<div></div>	0	🔍 Verified	<div></div>	0
⚪ Not-applicable	<div></div>	0	⚪ Not-applicable	<div></div>	0	⚪ Not-applicable	<div></div>	0
🟢 Recommended	<div></div>	3	🟢 Recommended	<div></div>	3	🟢 Recommended	<div></div>	3
❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0
⚪ Non-compliant	<div></div>	3	⚪ Non-compliant	<div></div>	3	⚪ Non-compliant	<div></div>	3
PCI-DSS-v3.2.1: 10.6			PCI-DSS-v3.2.1: 10.7			PCI-DSS-v3.2.1: 10.8		
✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0
🟡 Required	<div></div>	0	🟡 Required	<div></div>	0	🟡 Required	<div></div>	0
❌ Failed	<div></div>	0	❌ Failed	<div></div>	0	❌ Failed	<div></div>	0
🔍 Verified	<div></div>	0	🔍 Verified	<div></div>	0	🔍 Verified	<div></div>	0
⚪ Not-applicable	<div></div>	0	⚪ Not-applicable	<div></div>	0	⚪ Not-applicable	<div></div>	0
🟢 Recommended	<div></div>	7	🟢 Recommended	<div></div>	3	🟢 Recommended	<div></div>	5
❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0
⚪ Non-compliant	<div></div>	7	⚪ Non-compliant	<div></div>	3	⚪ Non-compliant	<div></div>	5
PCI-DSS-v3.2.1: 11.1			PCI-DSS-v3.2.1: 11.2			PCI-DSS-v3.2.1: 11.3		
✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0
🟡 Required	<div></div>	0	🟡 Required	<div></div>	0	🟡 Required	<div></div>	0
❌ Failed	<div></div>	0	❌ Failed	<div></div>	0	❌ Failed	<div></div>	0
🔍 Verified	<div></div>	0	🔍 Verified	<div></div>	0	🔍 Verified	<div></div>	0
⚪ Not-applicable	<div></div>	0	⚪ Not-applicable	<div></div>	0	⚪ Not-applicable	<div></div>	0
🟢 Recommended	<div></div>	5	🟢 Recommended	<div></div>	1	🟢 Recommended	<div></div>	3
❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0
⚪ Non-compliant	<div></div>	5	⚪ Non-compliant	<div></div>	1	⚪ Non-compliant	<div></div>	3

PCI-DSS-v3.2.1: 11.4			PCI-DSS-v3.2.1: 11.5			PCI-DSS-v3.2.1: 11.6		
✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0
⚠ Required	<div></div>	0	⚠ Required	<div></div>	0	⚠ Required	<div></div>	0
❌ Failed	<div></div>	0	❌ Failed	<div></div>	0	❌ Failed	<div></div>	0
✅ Verified	<div></div>	0	✅ Verified	<div></div>	0	✅ Verified	<div></div>	0
🛡 Not-applicable	<div></div>	0	🛡 Not-applicable	<div></div>	0	🛡 Not-applicable	<div></div>	0
🔒 Recommended	<div></div>	6	🔒 Recommended	<div></div>	12	🔒 Recommended	<div></div>	1
❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0
⚠ Non-compliant	<div></div>	6	⚠ Non-compliant	<div></div>	12	⚠ Non-compliant	<div></div>	1
PCI-DSS-v3.2.1: 12.1			PCI-DSS-v3.2.1: 12.10			PCI-DSS-v3.2.1: 12.2		
✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0
⚠ Required	<div></div>	0	⚠ Required	<div></div>	0	⚠ Required	<div></div>	0
❌ Failed	<div></div>	0	❌ Failed	<div></div>	0	❌ Failed	<div></div>	0
✅ Verified	<div></div>	0	✅ Verified	<div></div>	0	✅ Verified	<div></div>	0
🛡 Not-applicable	<div></div>	0	🛡 Not-applicable	<div></div>	0	🛡 Not-applicable	<div></div>	0
🔒 Recommended	<div></div>	1	🔒 Recommended	<div></div>	3	🔒 Recommended	<div></div>	2
❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0
⚠ Non-compliant	<div></div>	1	⚠ Non-compliant	<div></div>	3	⚠ Non-compliant	<div></div>	2
PCI-DSS-v3.2.1: 12.3			PCI-DSS-v3.2.1: 12.8					
✔ Implemented	<div></div>	0	✔ Implemented	<div></div>	0			
⚠ Required	<div></div>	0	⚠ Required	<div></div>	0			
❌ Failed	<div></div>	0	❌ Failed	<div></div>	0			
✅ Verified	<div></div>	0	✅ Verified	<div></div>	0			
🛡 Not-applicable	<div></div>	0	🛡 Not-applicable	<div></div>	0			
🔒 Recommended	<div></div>	2	🔒 Recommended	<div></div>	3			
❌ Rejected	<div></div>	0	❌ Rejected	<div></div>	0			
⚠ Non-compliant	<div></div>	2	⚠ Non-compliant	<div></div>	3			

Content menu

- PCI-DSS-v3.2.1: 1
 - Non-compliant
 - Recommended
- PCI-DSS-v3.2.1: 2
 - Non-compliant
 - Recommended
- PCI-DSS-v3.2.1: 3
 - Non-compliant
 - Recommended
- PCI-DSS-v3.2.1: 4
 - Non-compliant
 - Recommended
- PCI-DSS-v3.2.1: 10.6.1
 - Non-compliant
 - Recommended
- PCI-DSS-v3.2.1: 10.6.2
 - Non-compliant
 - Recommended
- PCI-DSS-v3.2.1: 10.6.3
 - Non-compliant
 - Recommended
- PCI-DSS-v3.2.1: 11.5.1
 - Non-compliant
 - Recommended
- PCI-DSS-v3.2.1: 12.10.5
 - Non-compliant
 - Recommended
- PCI-DSS-v3.2.1: 12.5.2
 - Non-compliant
 - Recommended
- PCI-DSS-v3.2.1: 9.1.1
 - Non-compliant
 - Recommended
- PCI-DSS-v3.2.1: 10.1
 - Non-compliant
 - Recommended
- PCI-DSS-v3.2.1: 10.2
 - Non-compliant
 - Recommended
- PCI-DSS-v3.2.1: 10.3
 - Non-compliant
 - Recommended
- PCI-DSS-v3.2.1: 10.4
 - Non-compliant
 - Recommended
- PCI-DSS-v3.2.1: 10.5
 - Non-compliant
 - Recommended
- PCI-DSS-v3.2.1: 10.6
 - Non-compliant
 - Recommended
- PCI-DSS-v3.2.1: 10.7
 - Non-compliant

Recommended
PCI-DSS-v3.2.1: 10.8
Non-compliant
Recommended
PCI-DSS-v3.2.1: 11.1
Non-compliant
Recommended
PCI-DSS-v3.2.1: 11.4
Non-compliant
Recommended
PCI-DSS-v3.2.1: 11.5
Non-compliant
Recommended
PCI-DSS-v3.2.1: 12.10
Non-compliant
Recommended
PCI-DSS-v3.2.1: 12.8
Non-compliant
Recommended
PCI-DSS-v3.2.1: 12.3.10
Non-compliant
Recommended
PCI-DSS-v3.2.1: 12.3.8
Non-compliant
Recommended
PCI-DSS-v3.2.1: 12.3.9
Non-compliant
Recommended
PCI-DSS-v3.2.1: 6.4.2
Non-compliant
Recommended
PCI-DSS-v3.2.1: 7.1.4
Non-compliant
Recommended
PCI-DSS-v3.2.1: 8.1.5
Non-compliant
Recommended
PCI-DSS-v3.2.1: 8.2.2
Non-compliant
Recommended
PCI-DSS-v3.2.1: 8.5.1
Non-compliant
Recommended
PCI-DSS-v3.2.1: 2.2
Non-compliant
Recommended
PCI-DSS-v3.2.1: 2.3
Non-compliant
Recommended
PCI-DSS-v3.2.1: 7.1
Non-compliant
Recommended
PCI-DSS-v3.2.1: 7.2
Non-compliant
Recommended

PCI-DSS-v3.2.1: 8.1
Non-compliant
Recommended
PCI-DSS-v3.2.1: 8.2
Non-compliant
Recommended
PCI-DSS-v3.2.1: 8.3
Non-compliant
Recommended
PCI-DSS-v3.2.1: 8.7
Non-compliant
Recommended
PCI-DSS-v3.2.1: 9.3
Non-compliant
Recommended
PCI-DSS-v3.2.1: 6.1
Non-compliant
Recommended
PCI-DSS-v3.2.1: 6.2
Non-compliant
Recommended
PCI-DSS-v3.2.1: 6.5
Non-compliant
Recommended
PCI-DSS-v3.2.1: 11.2
Non-compliant
Recommended
PCI-DSS-v3.2.1: 11.3
Non-compliant
Recommended
PCI-DSS-v3.2.1: 12.2
Non-compliant
Recommended
PCI-DSS-v3.2.1: 8.2.1
Non-compliant
Recommended
PCI-DSS-v3.2.1: 11.1.1
Non-compliant
Recommended
PCI-DSS-v3.2.1: 12.3.3
Non-compliant
Recommended
PCI-DSS-v3.2.1: 12.3.7
Non-compliant
Recommended
PCI-DSS-v3.2.1: 1.2
Non-compliant
Recommended
PCI-DSS-v3.2.1: 2.4
Non-compliant
Recommended
PCI-DSS-v3.2.1: 6.4
Non-compliant
Recommended

PCI-DSS-v3.2.1: 9.5
Non-compliant
Recommended

PCI-DSS-v3.2.1: 9.6
Non-compliant
Recommended

PCI-DSS-v3.2.1: 9.7
Non-compliant
Recommended

PCI-DSS-v3.2.1: 9.8
Non-compliant
Recommended

PCI-DSS-v3.2.1: 9.9
Non-compliant
Recommended

PCI-DSS-v3.2.1: 2.1
Non-compliant
Recommended

PCI-DSS-v3.2.1: 8.5
Non-compliant
Recommended

PCI-DSS-v3.2.1: 8.6
Non-compliant
Recommended

PCI-DSS-v3.2.1: 12.3
Non-compliant
Recommended

PCI-DSS-v3.2.1: 1.1.1
Non-compliant
Recommended

PCI-DSS-v3.2.1: 1.1.2
Non-compliant
Recommended

PCI-DSS-v3.2.1: 1.1.3
Non-compliant
Recommended

PCI-DSS-v3.2.1: 1.1
Non-compliant
Recommended

PCI-DSS-v3.2.1: 1.3
Non-compliant
Recommended

PCI-DSS-v3.2.1: 1.5
Non-compliant
Recommended

PCI-DSS-v3.2.1: 2.5
Non-compliant
Recommended

PCI-DSS-v3.2.1: 3.7
Non-compliant
Recommended

PCI-DSS-v3.2.1: 4.3
Non-compliant
Recommended

PCI-DSS-v3.2.1: 5.4

- Non-compliant
- Recommended
- PCI-DSS-v3.2.1: 6.7
- Non-compliant
- Recommended
- PCI-DSS-v3.2.1: 7.3
- Non-compliant
- Recommended
- PCI-DSS-v3.2.1: 8.8
- Non-compliant
- Recommended
- PCI-DSS-v3.2.1: 9.10
- Non-compliant
- Recommended
- PCI-DSS-v3.2.1: 11.6
- Non-compliant
- Recommended
- PCI-DSS-v3.2.1: 12.1
- Non-compliant
- Recommended
- PCI-DSS-v3.2.1: 6.4.1
- Non-compliant
- Recommended

PCI-DSS-v3.2.1: 1

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

2 Recommended countermeasures

Rec1. Block or validate all outbound requests

C-WEB-APPLICATION-SERVER-SIDE-CNT-11

Medium

Not tested

State: Recommended

Description:

To reduce the risk of data exfiltration or unauthorized system communication, block or carefully validate all outbound requests made by the application. Ensure that only requests to specific, trusted domains and IP addresses are allowed, and that any outbound request to unapproved destinations is blocked or flagged for further review. This control prevents the application from communicating with malicious external systems and protects sensitive data from being leaked.

Implementation Steps:

- Implement Outbound Request Filtering:** Use a web application firewall (WAF) or proxy to block or monitor any outbound requests that are not to trusted domains or IP addresses.
- Validate Outbound Requests:** Before allowing outbound communication, validate the destination domain and IP address to ensure they are within an approved list of resources necessary for the application.
- Set Domain and IP Allowlists:** Define an allowlist of specific domains and IP addresses that the application is authorized to interact with, and block all other outbound traffic.
- Monitor and Log Outbound Requests:** Continuously monitor and log all outbound requests for suspicious activity or attempts to communicate with unapproved destinations.
- Review and Update Allowlist Regularly:** Periodically review the allowlist to ensure that it remains up-to-date with the application's legitimate requirements.

References:

- [Security and Privacy Controls for Information Systems and Organizations](#)

Rec2. Use TLS for communications and protect stored data with encryption

C-WEB-APPLICATION-SERVER-SIDE-CNT-05

Very high

Not tested

State: Recommended

Description:

To secure data during transmission and at rest, ensure that all communications are protected using Transport Layer Security (TLS), and that stored data is encrypted using strong encryption algorithms. Additionally, implement robust key management practices to safeguard encryption keys and ensure that only authorized users and systems can access sensitive data.

Implementation Steps:

- Enforce TLS for All Communications:** Configure all communication channels, including APIs, web traffic, and data transfers, to use TLS (preferably TLS 1.2 or higher) to protect data in transit.
- Encrypt Stored Data:** Use strong encryption algorithms (e.g., AES-256) to encrypt sensitive data at rest, ensuring that unauthorized users cannot access or manipulate it.
- Implement Key Management Practices:** Use a centralized key management system (KMS) to securely generate, store, and rotate encryption keys. Enforce access controls to ensure that only authorized systems can access encryption keys.
- Regularly Update and Rotate Keys:** Set up automated key rotation policies to periodically update encryption keys and minimize the risk of key compromise.

References:

- [OWASP Transport Layer Security \(TLS\) Cheat Sheet](#)

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Report: CodeSentinel Thread Modeling

Compliance report - 2025-03-14T18:54:57.395420405Z

13 of 97

Component: Web Application - Server side

- Rec1. Block or validate all outbound requests C-WEB-APPLICATION-SERVER-SIDE-CNT-11

Medium
- Rec2. Use TLS for communications and protect stored data with encryption C-WEB-APPLICATION-SERVER-SIDE-CNT-05

Very high

PCI-DSS-v3.2.1: 2

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

2 Recommended countermeasures

Rec1. Block or validate all outbound requests C-WEB-APPLICATION-SERVER-SIDE-CNT-11

Medium

Not tested

State:

Recommended

Description:

To reduce the risk of data exfiltration or unauthorized system communication, block or carefully validate all outbound requests made by the application. Ensure that only requests to specific, trusted domains and IP addresses are allowed, and that any outbound request to unapproved destinations is blocked or flagged for further review. This control prevents the application from communicating with malicious external systems and protects sensitive data from being leaked.

Implementation Steps:

1. Implement Outbound Request Filtering:

Use a web application firewall (WAF) or proxy to block or monitor any outbound requests that are not to trusted domains or IP addresses.

2. Validate Outbound Requests:

Before allowing outbound communication, validate the destination domain and IP address to ensure they are within an approved list of resources necessary for the application.

3. Set Domain and IP Allowlists:

Define an allowlist of specific domains and IP addresses that the application is authorized to interact with, and block all other outbound traffic.

4. Monitor and Log Outbound Requests:

Continuously monitor and log all outbound requests for suspicious activity or attempts to communicate with unapproved destinations.

5. Review and Update Allowlist Regularly:

Periodically review the allowlist to ensure that it remains up-to-date with the application's legitimate requirements.

References:

Security and Privacy Controls for Information Systems and Organizations

Rec2. Use TLS for communications and protect stored data with encryption C-WEB-APPLICATION-SERVER-SIDE-CNT-05

Very high

Not tested

State:

Recommended

Description:

To secure data during transmission and at rest, ensure that all communications are protected using Transport Layer Security (TLS), and that stored data is encrypted using strong encryption algorithms. Additionally, implement robust key management practices to safeguard encryption keys and ensure that only authorized users and systems can access sensitive data.

Implementation Steps:

1. Enforce TLS for All Communications:

Configure all communication channels, including APIs, web traffic, and data transfers, to use TLS (preferably TLS 1.2 or higher) to protect data in transit.

2. Encrypt Stored Data:

Use strong encryption algorithms (e.g., AES-256) to encrypt sensitive data at rest, ensuring that unauthorized users cannot access or manipulate it.

3. Implement Key Management Practices:

Use a centralized key management system (KMS) to securely generate, store, and rotate encryption keys. Enforce access controls to ensure that only authorized systems can access encryption keys.

4. Regularly Update and Rotate Keys:

Set up automated key rotation policies to periodically update encryption keys and minimize the risk of key compromise.

References:

OWASP Transport Layer Security (TLS) Cheat Sheet

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Block or validate all outbound requests C-WEB-APPLICATION-SERVER-SIDE-CNT-11

Medium

Rec2. Use TLS for communications and protect stored data with encryption C-WEB-APPLICATION-SERVER-SIDE-CNT-05

Very high

PCI-DSS-v3.2.1: 3

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Document-Oriented Database

2 Recommended countermeasures

Rec1. Data Masking C-DOCUMENT-ORIENTED-DATABASE-CM4

Very high

Not tested

State:

Recommended

Description:

Data masking is a critical security measure for protecting sensitive information stored in a document-oriented database. It involves obscuring specific data elements within the database to prevent unauthorized users from viewing the actual content. Here's how to implement data masking effectively:

1. Identify Sensitive Data

Catalog Sensitive Information:

Conduct a thorough assessment of your database to identify sensitive data that requires masking. This may include personal identifiable information (PII), financial details, health records, or other confidential information.

Classify Data:

Categorize the data based on sensitivity levels to determine the appropriate masking techniques for each category.

2. Choose Masking Techniques

Static Masking:

Apply static data masking for non-production environments. Replace sensitive data with realistic, but not real, data suitable for development, testing, or training.

Dynamic Masking:

Implement dynamic data masking for production environments. It allows data to be masked on-the-fly, displaying masked data to unauthorized users while keeping the actual data intact and accessible for authorized users.

De-identification:

Remove or modify personal identifiers to protect personal privacy. This is crucial for compliance with data protection regulations.


Report: CodeSentinel Thread Modeling

Compliance report - 2025-03-14T18:54:57.395420405Z

14 of 97

- **Tokenization:** Use tokenization for highly sensitive data. Replace data elements with non-sensitive equivalents, known as tokens, which can be mapped back to the original data only with a specific key.
3. Implement Data Masking
- **Use Database Management Tools:** Leverage data masking features provided by your document-oriented database management system or third-party tools designed for data masking.
 - **Automate Masking Processes:** Where possible, automate the data masking process to ensure consistency and efficiency. This is particularly important in dynamic environments or when dealing with large volumes of data.
 - **Mask Data in Transit:** Ensure that data masking also applies to data in transit between the database and application layers to prevent exposure through interception.
4. Test Masked Data
- **Verify Masking Logic:** Test the masking implementation to ensure that the data is appropriately masked based on user roles and permissions. Confirm that masked data remains realistic and usable for its intended purpose, especially in non-production environments.
 - **Performance Impact Analysis:** Assess the performance impact of data masking on your database operations and application response times. Adjust the masking strategy as needed to balance security with performance.
5. Monitor and Audit
- **Access Monitoring:** Continuously monitor access to sensitive data and the effectiveness of the masking controls. Look for unauthorized attempts to access or unmask the data.
 - **Regular Audits:** Conduct regular audits of the data masking implementation to ensure it aligns with evolving data protection regulations and organizational policies.
6. Update Masking Rules as Needed
- **Stay Current:** Keep abreast of new data protection laws and security threats. Update your data masking rules and techniques accordingly to address new requirements or vulnerabilities.
- References and Good Security Practices**
- **Follow Best Practices:** Adhere to best practices for data masking laid out by organizations such as the Open Web Application Security Project (OWASP) and the National Institute of Standards and Technology (NIST).
 - **Data Protection Regulations:** Ensure your data masking strategy complies with relevant data protection regulations like GDPR, CCPA, HIPAA, etc.
- Implementing data masking in a document-oriented database helps minimize the risk of sensitive data exposure, ensuring that confidential information remains secure even in the event of unauthorized access.

Rec2. Encryption at Rest C-DOCUMENT-ORIENTED-DATABASE-CM3 Very high ☐ Not tested

- State:  Recommended
- Description:

To enhance the security of your document-oriented database by protecting stored data against unauthorized access, implement encryption at rest. This involves encrypting the data files on the server so that they cannot be read without the encryption key, safeguarding sensitive information even if physical access to the storage is obtained. Follow these steps to apply encryption at rest effectively:

Step 1: Choose an Encryption Method

Research Encryption Algorithms: Opt for strong, industry-standard encryption algorithms like AES-256 to secure your data at rest. Ensure the encryption solution supports your chosen algorithm.

Select an Encryption Solution: Determine whether your document-oriented database offers native encryption at rest features. If not, consider third-party encryption tools or filesystem-level encryption solutions compatible with your operating system.

Step 2: Implement Encryption

Native Database Encryption:

 - If your database system supports native encryption at rest, enable this feature according to the database documentation. This might involve configuring database settings and restarting the database service.
 - Set up key management according to best practices, which may involve using a secure key management system (KMS) or hardware security module (HSM).

Filesystem-Level Encryption:

 - For databases without native encryption, use filesystem-level encryption tools such as dm-crypt for Linux or BitLocker for Windows. Configure these tools to encrypt the volume where your database files are stored.
 - Initialize the encryption tool, create an encrypted volume, and move your database files to this volume, following the tool's documentation.

Step 3: Manage Encryption Keys

Secure Key Storage: Store encryption keys securely, separate from the encrypted data. Use a dedicated key management service (KMS) when possible.

Key Rotation: Regularly rotate encryption keys to enhance security. Ensure your encryption solution or database system supports key rotation without data loss.

Access Controls for Keys: Implement strict access controls for encryption key management, allowing only authorized personnel to manage or access the encryption keys.

Step 4: Regular Testing and Auditing

Test Encryption Implementation: Regularly test the encryption setup to ensure data is properly encrypted and that keys are not exposed. This could involve auditing file access paths and attempting to access data without the encryption keys.

Audit Access Logs: Monitor and audit access logs for any unauthorized attempts to access encrypted data or encryption keys.

Step 5: Backup and Recovery

Encrypt Backups: Ensure that backups of your database are also encrypted at rest. Apply the same encryption standards to backups as you do to live data.

Test Recovery Processes: Regularly test backup recovery processes to ensure that encrypted data can be successfully restored and decrypted with the appropriate keys.

Additional Good Security Practices

 - **Document Procedures:** Maintain detailed documentation of your encryption protocols, key management processes, and emergency procedures for key compromise situations.
 - **Compliance and Legal Considerations:** Ensure your encryption at rest strategy complies with relevant data protection regulations and industry standards.
 - **Educate Your Team:** Provide training for your team on the importance of encryption at rest and secure key management practices.

By following these steps, you can effectively implement encryption at rest for your document-oriented database, significantly enhancing the security of stored data against unauthorized access or breaches.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Document-Oriented Database

- Rec1. Data Masking C-DOCUMENT-ORIENTED-DATABASE-CM4 Very high
- Rec2. Encryption at Rest C-DOCUMENT-ORIENTED-DATABASE-CM3 Very high

PCI-DSS-v3.2.1: 4


Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

1 Recommended countermeasures

Rec1. Use TLS for communications and protect stored data with encryption C-WEB-APPLICATION-SERVER-SIDE-CNT-05 Very high ☐ Not tested

- State:  Recommended
- Description:

To secure data during transmission and at rest, ensure that all communications are protected using Transport Layer Security (TLS), and that stored data is encrypted using strong encryption algorithms. Additionally, implement robust key management practices to safeguard encryption keys and ensure that only authorized users and systems can access sensitive data.

Implementation Steps:

 1. **Enforce TLS for All Communications:** Configure all communication channels, including APIs, web traffic, and data transfers, to use TLS (preferably TLS 1.2 or higher) to protect data in transit.
 2. **Encrypt Stored Data:** Use strong encryption algorithms (e.g., AES-256) to encrypt sensitive data at rest, ensuring that unauthorized users cannot access or manipulate it.
 3. **Implement Key Management Practices:** Use a centralized key management system (KMS) to securely generate, store, and rotate encryption keys. Enforce access controls to ensure that only authorized systems can access encryption keys.

4. **Regularly Update and Rotate Keys:** Set up automated key rotation policies to periodically update encryption keys and minimize the risk of key compromise.

References:

- OWASP Transport Layer Security (TLS) Cheat Sheet

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Use TLS for communications and protect stored data with encryption C-WEB-APPLICATION-SERVER-SIDE-CNT-05 Very high

PCI-DSS-v3.2.1: 10.6.1

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

4 Recommended countermeasures

Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High Not tested

- State: Recommended
- Description:

To minimize the attack surface and enhance security, disable any unused services, remove default accounts, and apply security updates promptly. By doing so, you reduce the risk of exploitation through unneeded services or default credentials and ensure that the system is protected from known vulnerabilities. Additionally, use secure configurations to prevent unauthorized access and ensure that the system is optimally protected.

Implementation Steps:

 - Disable Unused Services:** Identify and disable any services that are not needed for the application or system to function. This limits the number of potential attack vectors.
 - Remove Default Accounts:** Remove or disable default accounts and ensure that all active accounts are assigned strong, unique credentials. Avoid using default settings that might be easily guessed by attackers.
 - Apply Security Updates:** Regularly check for and apply security patches to all systems and applications. Set up an automated patch management process to ensure that critical updates are applied as soon as they are released.
 - Use Secure Configurations:** Ensure that all configurations, such as database, server, and application settings, follow security best practices, including strong encryption, least privilege access, and secure communication protocols.
- References:
 - OWASP Secure Configuration Guide

Rec2. Block or validate all outbound requests C-WEB-APPLICATION-SERVER-SIDE-CNT-11 Medium Not tested

- State: Recommended
- Description:

To reduce the risk of data exfiltration or unauthorized system communication, block or carefully validate all outbound requests made by the application. Ensure that only requests to specific, trusted domains and IP addresses are allowed, and that any outbound request to unapproved destinations is blocked or flagged for further review. This control prevents the application from communicating with malicious external systems and protects sensitive data from being leaked.

Implementation Steps:

 - Implement Outbound Request Filtering:** Use a web application firewall (WAF) or proxy to block or monitor any outbound requests that are not to trusted domains or IP addresses.
 - Validate Outbound Requests:** Before allowing outbound communication, validate the destination domain and IP address to ensure they are within an approved list of resources necessary for the application.
 - Set Domain and IP Allowlists:** Define an allowlist of specific domains and IP addresses that the application is authorized to interact with, and block all other outbound traffic.
 - Monitor and Log Outbound Requests:** Continuously monitor and log all outbound requests for suspicious activity or attempts to communicate with unapproved destinations.
 - Review and Update Allowlist Regularly:** Periodically review the allowlist to ensure that it remains up-to-date with the application's legitimate requirements.
- References:
 - Security and Privacy Controls for Information Systems and Organizations

Rec3. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09 Medium Not tested

- State: Recommended
- Description:

To ensure comprehensive visibility into system activities, capture detailed logs for all critical actions within the web application. These logs should include data such as user actions, system changes, and access to sensitive resources. Additionally, proactively monitor these logs to detect suspicious behavior, unauthorized access, or anomalous activity, enabling timely detection of security incidents and faster response to potential threats.

Implementation Steps:

 - Enable Logging for Critical Actions:** Ensure that all key actions (e.g., login attempts, data access, configuration changes) are logged in detail, including relevant metadata such as timestamps, user identifiers, and source IP addresses.
 - Use Centralized Logging:** Store logs in a centralized location for easier access and analysis. Integrate with a log management or SIEM system for real-time monitoring.
 - Set Up Automated Alerts:** Configure automated alerts for suspicious or unauthorized actions, such as failed login attempts, unexpected configuration changes, or access to sensitive data by unauthorized users.
 - Regularly Review Logs:** Conduct regular log reviews and audits to ensure that logging mechanisms are functioning correctly and that logs provide the necessary level of detail to identify security incidents.
- References:
 - OWASP Logging Best Practices

Rec4. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12 Very high Not tested

- State: Recommended
- Description:

To prevent workflow abuses, especially for critical or sensitive transactions, implement server-side checks and multi-step validation. This ensures that transactions are properly validated at multiple points in the process, reducing the risk of unauthorized actions, fraud, or exploitation of business workflows.

Implementation Steps:

 - Define Critical Transactions:** Identify which transactions are critical (e.g., financial transactions, user permissions changes) and require extra validation to ensure security.
 - Server-Side Validation:** Ensure that all business rules, permissions, and workflows are validated on the server side, rather than relying solely on client-side validation.
 - Multi-Step Validation Process:** Implement a multi-step validation process for important transactions, requiring approval or confirmation at multiple stages (e.g., email confirmation, admin approval, CAPTCHA verification).

4. **Audit Trail:** Maintain a detailed audit trail for critical transactions, logging every step and any changes made, so that any suspicious activity can be detected and reviewed.
5. **Review and Update Validation Rules Regularly:** Periodically review and update the validation rules to adapt to new business requirements and emerging threats.

References:

- [PCI DSS Quick Reference Guide](#)

Component: API Service

1 Recommended countermeasures

Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high ○ Not tested

- State: ● Recommended
- Description:

In order to fortify the security posture of our API Service, it is imperative to implement a robust logging mechanism and seamlessly integrate it with a Security Information and Event Management (SIEM) system. This control ensures that all relevant activities and events within the API ecosystem are recorded, monitored, and analyzed in real-time, enhancing our ability to detect and respond to security incidents promptly.

Implementation Guidelines:

Granular Logging:

 - Implement detailed and granular logging mechanisms to capture pertinent information about API requests, responses, user authentication, and authorization events.
 - Log relevant metadata such as timestamps, source IP addresses, user identities, and the nature of the API transactions.

Sensitive Data Masking:

 - Apply appropriate data masking techniques to prevent the logging of sensitive information such as passwords, personal identifiable information (PII), and other confidential data.

Logging Standards:

 - Adhere to industry-standard logging formats (e.g., JSON, syslog) to ensure compatibility with SIEM solutions and facilitate efficient log parsing and analysis.

Log Retention and Rotation:

 - Define a log retention policy to ensure logs are retained for an appropriate duration based on compliance requirements and security best practices.
 - Implement log rotation to prevent storage overflow and maintain a manageable log volume.

Real-time Alerting:

 - Integrate the logging system with a SIEM solution to enable real-time monitoring and alerting on suspicious or anomalous activities.
 - Configure alerts for specific events that may indicate security incidents, such as multiple failed login attempts or unusual patterns of API access.

Access Controls for Logs:

 - Implement access controls and encryption for log files to restrict access only to authorized personnel.
 - Regularly review and audit access to log files to ensure compliance and detect unauthorized access.

Integration with Incident Response:

 - Ensure that the logging system is seamlessly integrated with the organization's incident response processes and tools.
 - Define clear procedures for incident identification, analysis, and response based on the information gathered from the logs.

Regular Monitoring and Review:

 - Periodically review and analyze logs to identify trends, potential security threats, or areas for improvement in the API service's security posture.
 - Conduct regular audits of the logging and SIEM integration to validate the effectiveness of the control.

Benefits:

 - Early detection of security incidents.
 - Improved forensic capabilities for investigating security breaches.
 - Compliance with regulatory requirements regarding data protection and incident reporting.
 - Enhanced visibility into API activities for both security and operational purposes.

Note to Developers: Implementing comprehensive logging and SIEM integration is not just a security requirement but a crucial aspect of maintaining the integrity and reliability of our API Service. Follow the guidelines diligently and collaborate with the security team to ensure the effective implementation of this control.

Component: Document-Oriented Database

1 Recommended countermeasures

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CMS Very high ○ Not tested

- State: ● Recommended
- Description:

Conducting regular configuration audits of a Document-Oriented Database is vital for ensuring that security settings and configurations are aligned with best practices and organizational security policies. Here's how to systematically approach these audits:

 - Establish a Baseline Configuration
 - **Identify Security Settings:** Document the optimal security configurations for your database environment. This includes access controls, encryption settings, network configurations, and any specific database security features.
 - **Create a Configuration Template:** Develop a template or checklist that outlines all security-related configuration settings. This template will serve as the baseline for regular audits.
 - Automate the Audit Process
 - **Use Configuration Management Tools:** Implement tools that can automate the process of detecting deviations from the baseline configuration. Tools like Puppet, Chef, Ansible, or database-specific management solutions can help.
 - **Schedule Regular Scans:** Configure these tools to perform regular scans of your document-oriented database environment to identify and report any configuration changes or non-compliance with the baseline.
 - Review Access Controls
 - **Validate User Access:** Regularly review user accounts, roles, and permissions to ensure that they adhere to the principle of least privilege. Remove or adjust any excessive permissions.
 - **Audit Authentication Mechanisms:** Verify that authentication methods (password policies, multi-factor authentication, etc.) are properly configured and enforced.
 - Verify Network Configurations
 - **Check Network Access Controls:** Ensure that the database is not exposed to unauthorized networks. Validate firewall rules and other network security configurations that control access to the database.
 - **Secure Communication Channels:** Confirm that data in transit is encrypted and that secure communication protocols are in use.
 - Confirm Data Protection Measures
 - **Data Encryption:** Verify that data at rest is encrypted according to best practices. Check the configuration of encryption mechanisms and the management of encryption keys.
 - **Backup and Recovery:** Review the configuration of backup processes to ensure that data is securely backed up and that encryption is maintained for backups.
 - Document and Resolve Findings
 - **Report Findings:** Document any deviations from the baseline configuration identified during the audit. Classify findings based on their potential impact on security.
 - **Plan Remediation:** Develop a plan for addressing high-priority findings. This may include reconfiguring settings, updating software, or modifying access controls.
 - **Track Progress:** Maintain records of audits, findings, and remediation efforts to track improvements over time and to provide accountability.
 - Update Baseline Configurations
 - **Incorporate Changes:** As you make changes to your database environment or as new security best practices emerge, update your baseline configuration and templates accordingly.
 - **Review Periodically:** Regularly review the baseline configuration to ensure it remains relevant and effective in mitigating security risks.
 - Educate and Train
 - **Team Awareness:** Ensure that your team is aware of the configuration audit process, the importance of adhering to the baseline configurations, and the role they play in maintaining database security.
 - **Continuous Learning:** Stay informed about new security features offered by your document-oriented database system and incorporate relevant features into your security baseline.

References and Good Security Practices

 - **Industry Best Practices:** Follow guidelines and security checklists provided by the database vendor, as well as security standards relevant to your industry (e.g., CIS Benchmarks, NIST Guidelines).
 - **Security Communities:** Engage with security communities and forums related to your database technology to stay updated on the latest security advisories and recommendations.

Regular configuration audits are a proactive step in maintaining the security and integrity of your document-oriented database, helping to prevent unauthorized access, data leaks, and ensuring compliance with data protection regulations.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

- Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High
- Rec2. Block or validate all outbound requests C-WEB-APPLICATION-SERVER-SIDE-CNT-11 Medium
- Rec3. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09 Medium
- Rec4. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12 Very high

Component: API Service

- Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high

Component: Document-Oriented Database

- Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CMS Very high

PCI-DSS-v3.2.1: 10.6.2

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

4 Recommended countermeasures

Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High Not tested

State: Recommended

Description:

To minimize the attack surface and enhance security, disable any unused services, remove default accounts, and apply security updates promptly. By doing so, you reduce the risk of exploitation through unneeded services or default credentials and ensure that the system is protected from known vulnerabilities. Additionally, use secure configurations to prevent unauthorized access and ensure that the system is optimally protected.

Implementation Steps:

- 1. Disable Unused Services:** Identify and disable any services that are not needed for the application or system to function. This limits the number of potential attack vectors.
- 2. Remove Default Accounts:** Remove or disable default accounts and ensure that all active accounts are assigned strong, unique credentials. Avoid using default settings that might be easily guessed by attackers.
- 3. Apply Security Updates:** Regularly check for and apply security patches to all systems and applications. Set up an automated patch management process to ensure that critical updates are applied as soon as they are released.
- 4. Use Secure Configurations:** Ensure that all configurations, such as database, server, and application settings, follow security best practices, including strong encryption, least privilege access, and secure communication protocols.

References:

- [OWASP Secure Configuration Guide](#)

Rec2. Block or validate all outbound requests C-WEB-APPLICATION-SERVER-SIDE-CNT-11 Medium Not tested

State: Recommended

Description:

To reduce the risk of data exfiltration or unauthorized system communication, block or carefully validate all outbound requests made by the application. Ensure that only requests to specific, trusted domains and IP addresses are allowed, and that any outbound request to unapproved destinations is blocked or flagged for further review. This control prevents the application from communicating with malicious external systems and protects sensitive data from being leaked.

Implementation Steps:

- 1. Implement Outbound Request Filtering:** Use a web application firewall (WAF) or proxy to block or monitor any outbound requests that are not to trusted domains or IP addresses.
- 2. Validate Outbound Requests:** Before allowing outbound communication, validate the destination domain and IP address to ensure they are within an approved list of resources necessary for the application.
- 3. Set Domain and IP Allowlists:** Define an allowlist of specific domains and IP addresses that the application is authorized to interact with, and block all other outbound traffic.
- 4. Monitor and Log Outbound Requests:** Continuously monitor and log all outbound requests for suspicious activity or attempts to communicate with unapproved destinations.
- 5. Review and Update Allowlist Regularly:** Periodically review the allowlist to ensure that it remains up-to-date with the application's legitimate requirements.

References:

- [Security and Privacy Controls for Information Systems and Organizations](#)

Rec3. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09 Medium Not tested

State: Recommended

Description:

To ensure comprehensive visibility into system activities, capture detailed logs for all critical actions within the web application. These logs should include data such as user actions, system changes, and access to sensitive resources. Additionally, proactively monitor these logs to detect suspicious behavior, unauthorized access, or anomalous activity, enabling timely detection of security incidents and faster response to potential threats.

Implementation Steps:

- 1. Enable Logging for Critical Actions:** Ensure that all key actions (e.g., login attempts, data access, configuration changes) are logged in detail, including relevant metadata such as timestamps, user identifiers, and source IP addresses.
- 2. Use Centralized Logging:** Store logs in a centralized location for easier access and analysis. Integrate with a log management or SIEM system for real-time monitoring.
- 3. Set Up Automated Alerts:** Configure automated alerts for suspicious or unauthorized actions, such as failed login attempts, unexpected configuration changes, or access to sensitive data by unauthorized users.
- 4. Regularly Review Logs:** Conduct regular log reviews and audits to ensure that logging mechanisms are functioning correctly and that logs provide the necessary level of detail to identify security incidents.

References:

- [OWASP Logging Best Practices](#)

Rec4. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12 Very high Not tested

State: Recommended

Description:

To prevent workflow abuses, especially for critical or sensitive transactions, implement server-side checks and multi-step validation. This ensures that transactions are properly validated at multiple points in the process, reducing the risk of unauthorized actions, fraud, or exploitation of business workflows.

Report: CodeSentinel Thread Modeling

Compliance report - 2025-03-14T18:54:57.395420405Z

18 of 97

Implementation Steps:

- 1. **Define Critical Transactions:** Identify which transactions are critical (e.g., financial transactions, user permissions changes) and require extra validation to ensure security.
- 2. **Server-Side Validation:** Ensure that all business rules, permissions, and workflows are validated on the server side, rather than relying solely on client-side validation.
- 3. **Multi-Step Validation Process:** Implement a multi-step validation process for important transactions, requiring approval or confirmation at multiple stages (e.g., email confirmation, admin approval, CAPTCHA verification).
- 4. **Audit Trail:** Maintain a detailed audit trail for critical transactions, logging every step and any changes made, so that any suspicious activity can be detected and reviewed.
- 5. **Review and Update Validation Rules Regularly:** Periodically review and update the validation rules to adapt to new business requirements and emerging threats.

References:

- [PCI DSS Quick Reference Guide](#)

Component: API Service

1 Recommended countermeasures

Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high ☐ Not tested

- State: ☒ Recommended
- Description:

In order to fortify the security posture of our API Service, it is imperative to implement a robust logging mechanism and seamlessly integrate it with a Security Information and Event Management (SIEM) system. This control ensures that all relevant activities and events within the API ecosystem are recorded, monitored, and analyzed in real-time, enhancing our ability to detect and respond to security incidents promptly.

Implementation Guidelines:

Granular Logging:

 - Implement detailed and granular logging mechanisms to capture pertinent information about API requests, responses, user authentication, and authorization events.
 - Log relevant metadata such as timestamps, source IP addresses, user identities, and the nature of the API transactions.

Sensitive Data Masking:

 - Apply appropriate data masking techniques to prevent the logging of sensitive information such as passwords, personal identifiable information (PII), and other confidential data.

Logging Standards:

 - Adhere to industry-standard logging formats (e.g., JSON, syslog) to ensure compatibility with SIEM solutions and facilitate efficient log parsing and analysis.

Log Retention and Rotation:

 - Define a log retention policy to ensure logs are retained for an appropriate duration based on compliance requirements and security best practices.
 - Implement log rotation to prevent storage overflow and maintain a manageable log volume.

Real-time Alerting:

 - Integrate the logging system with a SIEM solution to enable real-time monitoring and alerting on suspicious or anomalous activities.
 - Configure alerts for specific events that may indicate security incidents, such as multiple failed login attempts or unusual patterns of API access.

Access Controls for Logs:

 - Implement access controls and encryption for log files to restrict access only to authorized personnel.
 - Regularly review and audit access to log files to ensure compliance and detect unauthorized access.

Integration with Incident Response:

 - Ensure that the logging system is seamlessly integrated with the organization's incident response processes and tools.
 - Define clear procedures for incident identification, analysis, and response based on the information gathered from the logs.

Regular Monitoring and Review:

 - Periodically review and analyze logs to identify trends, potential security threats, or areas for improvement in the API service's security posture.
 - Conduct regular audits of the logging and SIEM integration to validate the effectiveness of the control.

Benefits:

 - Early detection of security incidents.
 - Improved forensic capabilities for investigating security breaches.
 - Compliance with regulatory requirements regarding data protection and incident reporting.
 - Enhanced visibility into API activities for both security and operational purposes.

Note to Developers: Implementing comprehensive logging and SIEM integration is not just a security requirement but a crucial aspect of maintaining the integrity and reliability of our API Service. Follow the guidelines diligently and collaborate with the security team to ensure the effective implementation of this control.

Component: Document-Oriented Database

1 Recommended countermeasures

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CMS Very high ☐ Not tested

- State: ☒ Recommended
- Description:

Conducting regular configuration audits of a Document-Oriented Database is vital for ensuring that security settings and configurations are aligned with best practices and organizational security policies. Here's how to systematically approach these audits:

 - Establish a Baseline Configuration
 - **Identify Security Settings:** Document the optimal security configurations for your database environment. This includes access controls, encryption settings, network configurations, and any specific database security features.
 - **Create a Configuration Template:** Develop a template or checklist that outlines all security-related configuration settings. This template will serve as the baseline for regular audits.
 - Automate the Audit Process
 - **Use Configuration Management Tools:** Implement tools that can automate the process of detecting deviations from the baseline configuration. Tools like Puppet, Chef, Ansible, or database-specific management solutions can help.
 - **Schedule Regular Scans:** Configure these tools to perform regular scans of your document-oriented database environment to identify and report any configuration changes or non-compliance with the baseline.
 - Review Access Controls
 - **Validate User Access:** Regularly review user accounts, roles, and permissions to ensure that they adhere to the principle of least privilege. Remove or adjust any excessive permissions.
 - **Audit Authentication Mechanisms:** Verify that authentication methods (password policies, multi-factor authentication, etc.) are properly configured and enforced.
 - Verify Network Configurations
 - **Check Network Access Controls:** Ensure that the database is not exposed to unauthorized networks. Validate firewall rules and other network security configurations that control access to the database.
 - **Secure Communication Channels:** Confirm that data in transit is encrypted and that secure communication protocols are in use.
 - Confirm Data Protection Measures
 - **Data Encryption:** Verify that data at rest is encrypted according to best practices. Check the configuration of encryption mechanisms and the management of encryption keys.
 - **Backup and Recovery:** Review the configuration of backup processes to ensure that data is securely backed up and that encryption is maintained for backups.
 - Document and Resolve Findings
 - **Report Findings:** Document any deviations from the baseline configuration identified during the audit. Classify findings based on their potential impact on security.
 - **Plan Remediation:** Develop a plan for addressing high-priority findings. This may include reconfiguring settings, updating software, or modifying access controls.
 - **Track Progress:** Maintain records of audits, findings, and remediation efforts to track improvements over time and to provide accountability.
 - Update Baseline Configurations
 - **Incorporate Changes:** As you make changes to your database environment or as new security best practices emerge, update your baseline configuration and templates accordingly.
 - **Review Periodically:** Regularly review the baseline configuration to ensure it remains relevant and effective in mitigating security risks.
 - Educate and Train
 - **Team Awareness:** Ensure that your team is aware of the configuration audit process, the importance of adhering to the baseline configurations, and the role they play in maintaining database security.
 - **Continuous Learning:** Stay informed about new security features offered by your document-oriented database system and incorporate relevant features into your security baseline.

References and Good Security Practices

 - **Industry Best Practices:** Follow guidelines and security checklists provided by the database vendor, as well as security standards relevant to your industry (e.g., CIS Benchmarks, NIST Guidelines).
 - **Security Communities:** Engage with security communities and forums related to your database technology to stay updated on the latest security advisories and recommendations.

Regular configuration audits are a proactive step in maintaining the security and integrity of your document-oriented database, helping to prevent unauthorized access, data leaks, and ensuring compliance with data protection regulations.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

- Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High
- Rec2. Block or validate all outbound requests C-WEB-APPLICATION-SERVER-SIDE-CNT-11 Medium
- Rec3. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09 Medium
- Rec4. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12 Very high

Component: API Service

- Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high

Component: Document-Oriented Database

- Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CMS Very high

PCI-DSS-v3.2.1: 10.6.3

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

2 Recommended countermeasures

- Rec1. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09 Medium Not tested

State: Recommended

Description:

To ensure comprehensive visibility into system activities, capture detailed logs for all critical actions within the web application. These logs should include data such as user actions, system changes, and access to sensitive resources. Additionally, proactively monitor these logs to detect suspicious behavior, unauthorized access, or anomalous activity, enabling timely detection of security incidents and faster response to potential threats.

Implementation Steps:

 - Enable Logging for Critical Actions:** Ensure that all key actions (e.g., login attempts, data access, configuration changes) are logged in detail, including relevant metadata such as timestamps, user identifiers, and source IP addresses.
 - Use Centralized Logging:** Store logs in a centralized location for easier access and analysis. Integrate with a log management or SIEM system for real-time monitoring.
 - Set Up Automated Alerts:** Configure automated alerts for suspicious or unauthorized actions, such as failed login attempts, unexpected configuration changes, or access to sensitive data by unauthorized users.
 - Regularly Review Logs:** Conduct regular log reviews and audits to ensure that logging mechanisms are functioning correctly and that logs provide the necessary level of detail to identify security incidents.

References:

 - [OWASP Logging Best Practices](#)
- Rec2. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12 Very high Not tested

State: Recommended

Description:

To prevent workflow abuses, especially for critical or sensitive transactions, implement server-side checks and multi-step validation. This ensures that transactions are properly validated at multiple points in the process, reducing the risk of unauthorized actions, fraud, or exploitation of business workflows.

Implementation Steps:

 - Define Critical Transactions:** Identify which transactions are critical (e.g., financial transactions, user permissions changes) and require extra validation to ensure security.
 - Server-Side Validation:** Ensure that all business rules, permissions, and workflows are validated on the server side, rather than relying solely on client-side validation.
 - Multi-Step Validation Process:** Implement a multi-step validation process for important transactions, requiring approval or confirmation at multiple stages (e.g., email confirmation, admin approval, CAPTCHA verification).
 - Audit Trail:** Maintain a detailed audit trail for critical transactions, logging every step and any changes made, so that any suspicious activity can be detected and reviewed.
 - Review and Update Validation Rules Regularly:** Periodically review and update the validation rules to adapt to new business requirements and emerging threats.

References:

 - [PCI DSS Quick Reference Guide](#)

Component: API Service

1 Recommended countermeasures

- Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high Not tested

State: Recommended

Description:

In order to fortify the security posture of our API Service, it is imperative to implement a robust logging mechanism and seamlessly integrate it with a Security Information and Event Management (SIEM) system. This control ensures that all relevant activities and events within the API ecosystem are recorded, monitored, and analyzed in real-time, enhancing our ability to detect and respond to security incidents promptly.

Implementation Guidelines:

Granular Logging:

 - Implement detailed and granular logging mechanisms to capture pertinent information about API requests, responses, user authentication, and authorization events.
 - Log relevant metadata such as timestamps, source IP addresses, user identities, and the nature of the API transactions.

Sensitive Data Masking:

 - Apply appropriate data masking techniques to prevent the logging of sensitive information such as passwords, personal identifiable information (PII), and other confidential data.

Logging Standards:

 - Adhere to industry-standard logging formats (e.g., JSON, syslog) to ensure compatibility with SIEM solutions and facilitate efficient log parsing and analysis.

Log Retention and Rotation:
- Report: CodeSentinel Thread Modeling

Compliance report - 2025-03-14T18:54:57.395420405Z

20 of 97

- Define a log retention policy to ensure logs are retained for an appropriate duration based on compliance requirements and security best practices.
 - Implement log rotation to prevent storage overflow and maintain a manageable log volume.
- Real-time Alerting:**
- Integrate the logging system with a SIEM solution to enable real-time monitoring and alerting on suspicious or anomalous activities.
 - Configure alerts for specific events that may indicate security incidents, such as multiple failed login attempts or unusual patterns of API access.
- Access Controls for Logs:**
- Implement access controls and encryption for log files to restrict access only to authorized personnel.
 - Regularly review and audit access to log files to ensure compliance and detect unauthorized access.
- Integration with Incident Response:**
- Ensure that the logging system is seamlessly integrated with the organization's incident response processes and tools.
 - Define clear procedures for incident identification, analysis, and response based on the information gathered from the logs.
- Regular Monitoring and Review:**
- Periodically review and analyze logs to identify trends, potential security threats, or areas for improvement in the API service's security posture.
 - Conduct regular audits of the logging and SIEM integration to validate the effectiveness of the control.
- Benefits:**
- Early detection of security incidents.
 - Improved forensic capabilities for investigating security breaches.
 - Compliance with regulatory requirements regarding data protection and incident reporting.
 - Enhanced visibility into API activities for both security and operational purposes.
- Note to Developers:** Implementing comprehensive logging and SIEM integration is not just a security requirement but a crucial aspect of maintaining the integrity and reliability of our API Service. Follow the guidelines diligently and collaborate with the security team to ensure the effective implementation of this control.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

- Rec1. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09 Medium
- Rec2. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12 Very high

Component: API Service

- Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high

PCI-DSS-v3.2.1: 11.5.1

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

2 Recommended countermeasures

- Rec1. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09 Medium Not tested
- State: Recommended
 - Description:
To ensure comprehensive visibility into system activities, capture detailed logs for all critical actions within the web application. These logs should include data such as user actions, system changes, and access to sensitive resources. Additionally, proactively monitor these logs to detect suspicious behavior, unauthorized access, or anomalous activity, enabling timely detection of security incidents and faster response to potential threats.
- Implementation Steps:**
1. **Enable Logging for Critical Actions:** Ensure that all key actions (e.g., login attempts, data access, configuration changes) are logged in detail, including relevant metadata such as timestamps, user identifiers, and source IP addresses.
 2. **Use Centralized Logging:** Store logs in a centralized location for easier access and analysis. Integrate with a log management or SIEM system for real-time monitoring.
 3. **Set Up Automated Alerts:** Configure automated alerts for suspicious or unauthorized actions, such as failed login attempts, unexpected configuration changes, or access to sensitive data by unauthorized users.
 4. **Regularly Review Logs:** Conduct regular log reviews and audits to ensure that logging mechanisms are functioning correctly and that logs provide the necessary level of detail to identify security incidents.
- References:**
- [OWASP Logging Best Practices](#)
- Rec2. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12 Very high Not tested
- State: Recommended
 - Description:
To prevent workflow abuses, especially for critical or sensitive transactions, implement server-side checks and multi-step validation. This ensures that transactions are properly validated at multiple points in the process, reducing the risk of unauthorized actions, fraud, or exploitation of business workflows.
- Implementation Steps:**
1. **Define Critical Transactions:** Identify which transactions are critical (e.g., financial transactions, user permissions changes) and require extra validation to ensure security.
 2. **Server-Side Validation:** Ensure that all business rules, permissions, and workflows are validated on the server side, rather than relying solely on client-side validation.
 3. **Multi-Step Validation Process:** Implement a multi-step validation process for important transactions, requiring approval or confirmation at multiple stages (e.g., email confirmation, admin approval, CAPTCHA verification).
 4. **Audit Trail:** Maintain a detailed audit trail for critical transactions, logging every step and any changes made, so that any suspicious activity can be detected and reviewed.
 5. **Review and Update Validation Rules Regularly:** Periodically review and update the validation rules to adapt to new business requirements and emerging threats.
- References:**
- [PCI DSS Quick Reference Guide](#)

Component: API Service

1 Recommended countermeasures

- Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high Not tested
- State: Recommended
 - Description:

In order to fortify the security posture of our API Service, it is imperative to implement a robust logging mechanism and seamlessly integrate it with a Security Information and Event Management (SIEM) system. This control ensures that all relevant activities and events within the API ecosystem are recorded, monitored, and analyzed in real-time, enhancing our ability to detect and respond to security incidents promptly.

Implementation Guidelines:

Granular Logging:

- Implement detailed and granular logging mechanisms to capture pertinent information about API requests, responses, user authentication, and authorization events.
- Log relevant metadata such as timestamps, source IP addresses, user identities, and the nature of the API transactions.

Sensitive Data Masking:

- Apply appropriate data masking techniques to prevent the logging of sensitive information such as passwords, personal identifiable information (PII), and other confidential data.

Logging Standards:

- Adhere to industry-standard logging formats (e.g., JSON, syslog) to ensure compatibility with SIEM solutions and facilitate efficient log parsing and analysis.

Log Retention and Rotation:

- Define a log retention policy to ensure logs are retained for an appropriate duration based on compliance requirements and security best practices.
- Implement log rotation to prevent storage overflow and maintain a manageable log volume.

Real-time Alerting:

- Integrate the logging system with a SIEM solution to enable real-time monitoring and alerting on suspicious or anomalous activities.
- Configure alerts for specific events that may indicate security incidents, such as multiple failed login attempts or unusual patterns of API access.

Access Controls for Logs:

- Implement access controls and encryption for log files to restrict access only to authorized personnel.
- Regularly review and audit access to log files to ensure compliance and detect unauthorized access.

Integration with Incident Response:

- Ensure that the logging system is seamlessly integrated with the organization's incident response processes and tools.
- Define clear procedures for incident identification, analysis, and response based on the information gathered from the logs.

Regular Monitoring and Review:

- Periodically review and analyze logs to identify trends, potential security threats, or areas for improvement in the API service's security posture.
- Conduct regular audits of the logging and SIEM integration to validate the effectiveness of the control.

Benefits:

- Early detection of security incidents.
- Improved forensic capabilities for investigating security breaches.
- Compliance with regulatory requirements regarding data protection and incident reporting.
- Enhanced visibility into API activities for both security and operational purposes.

Note to Developers: Implementing comprehensive logging and SIEM integration is not just a security requirement but a crucial aspect of maintaining the integrity and reliability of our API Service. Follow the guidelines diligently and collaborate with the security team to ensure the effective implementation of this control.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

- Rec1. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09

Medium
- Rec2. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12

Very high

Component: API Service

- Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM

Very high

PCI-DSS-v3.2.1: 12.10.5

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

3 Recommended countermeasures

- Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06

High

Not tested

State: Recommended

Description:

To minimize the attack surface and enhance security, disable any unused services, remove default accounts, and apply security updates promptly. By doing so, you reduce the risk of exploitation through unneeded services or default credentials and ensure that the system is protected from known vulnerabilities. Additionally, use secure configurations to prevent unauthorized access and ensure that the system is optimally protected.

Implementation Steps:

 - Disable Unused Services:** Identify and disable any services that are not needed for the application or system to function. This limits the number of potential attack vectors.
 - Remove Default Accounts:** Remove or disable default accounts and ensure that all active accounts are assigned strong, unique credentials. Avoid using default settings that might be easily guessed by attackers.
 - Apply Security Updates:** Regularly check for and apply security patches to all systems and applications. Set up an automated patch management process to ensure that critical updates are applied as soon as they are released.
 - Use Secure Configurations:** Ensure that all configurations, such as database, server, and application settings, follow security best practices, including strong encryption, least privilege access, and secure communication protocols.

References:

 - [OWASP Secure Configuration Guide](#)

Rec2. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09

Medium

Not tested

State: Recommended

Description:

To ensure comprehensive visibility into system activities, capture detailed logs for all critical actions within the web application. These logs should include data such as user actions, system changes, and access to sensitive resources. Additionally, proactively monitor these logs to detect suspicious behavior, unauthorized access, or anomalous activity, enabling timely detection of security incidents and faster response to potential threats.

Implementation Steps:

 - Enable Logging for Critical Actions:** Ensure that all key actions (e.g., login attempts, data access, configuration changes) are logged in detail, including relevant metadata such as timestamps, user identifiers, and source IP addresses.
 - Use Centralized Logging:** Store logs in a centralized location for easier access and analysis. Integrate with a log management or SIEM system for real-time monitoring.
 - Set Up Automated Alerts:** Configure automated alerts for suspicious or unauthorized actions, such as failed login attempts, unexpected configuration changes, or access to sensitive data by unauthorized users.
 - Regularly Review Logs:** Conduct regular log reviews and audits to ensure that logging mechanisms are functioning correctly and that logs provide the necessary level of detail to identify security incidents.

Report: CodeSentinel Thread Modeling

Compliance report - 2025-03-14T18:54:57.395420405Z

22 of 97


References:

- OWASP Logging Best Practices

Rec3. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12

Very high

☐ Not tested

State:  Recommended

Description:

To prevent workflow abuses, especially for critical or sensitive transactions, implement server-side checks and multi-step validation. This ensures that transactions are properly validated at multiple points in the process, reducing the risk of unauthorized actions, fraud, or exploitation of business workflows.

Implementation Steps:

- Define Critical Transactions:** Identify which transactions are critical (e.g., financial transactions, user permissions changes) and require extra validation to ensure security.
- Server-Side Validation:** Ensure that all business rules, permissions, and workflows are validated on the server side, rather than relying solely on client-side validation.
- Multi-Step Validation Process:** Implement a multi-step validation process for important transactions, requiring approval or confirmation at multiple stages (e.g., email confirmation, admin approval, CAPTCHA verification).
- Audit Trail:** Maintain a detailed audit trail for critical transactions, logging every step and any changes made, so that any suspicious activity can be detected and reviewed.
- Review and Update Validation Rules Regularly:** Periodically review and update the validation rules to adapt to new business requirements and emerging threats.

References:

- PCI DSS Quick Reference Guide

Component: API Service

1 Recommended countermeasures

Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM

Very high

☐ Not tested

State:  Recommended

Description:

In order to fortify the security posture of our API Service, it is imperative to implement a robust logging mechanism and seamlessly integrate it with a Security Information and Event Management (SIEM) system. This control ensures that all relevant activities and events within the API ecosystem are recorded, monitored, and analyzed in real-time, enhancing our ability to detect and respond to security incidents promptly.

Implementation Guidelines:

Granular Logging:

- Implement detailed and granular logging mechanisms to capture pertinent information about API requests, responses, user authentication, and authorization events.
- Log relevant metadata such as timestamps, source IP addresses, user identities, and the nature of the API transactions.

Sensitive Data Masking:

- Apply appropriate data masking techniques to prevent the logging of sensitive information such as passwords, personal identifiable information (PII), and other confidential data.

Logging Standards:

- Adhere to industry-standard logging formats (e.g., JSON, syslog) to ensure compatibility with SIEM solutions and facilitate efficient log parsing and analysis.

Log Retention and Rotation:

- Define a log retention policy to ensure logs are retained for an appropriate duration based on compliance requirements and security best practices.
- Implement log rotation to prevent storage overflow and maintain a manageable log volume.

Real-time Alerting:

- Integrate the logging system with a SIEM solution to enable real-time monitoring and alerting on suspicious or anomalous activities.
- Configure alerts for specific events that may indicate security incidents, such as multiple failed login attempts or unusual patterns of API access.

Access Controls for Logs:

- Implement access controls and encryption for log files to restrict access only to authorized personnel.
- Regularly review and audit access to log files to ensure compliance and detect unauthorized access.

Integration with Incident Response:

- Ensure that the logging system is seamlessly integrated with the organization's incident response processes and tools.
- Define clear procedures for incident identification, analysis, and response based on the information gathered from the logs.

Regular Monitoring and Review:

- Periodically review and analyze logs to identify trends, potential security threats, or areas for improvement in the API service's security posture.
- Conduct regular audits of the logging and SIEM integration to validate the effectiveness of the control.

Benefits:

- Early detection of security incidents.
- Improved forensic capabilities for investigating security breaches.
- Compliance with regulatory requirements regarding data protection and incident reporting.
- Enhanced visibility into API activities for both security and operational purposes.

Note to Developers: Implementing comprehensive logging and SIEM integration is not just a security requirement but a crucial aspect of maintaining the integrity and reliability of our API Service. Follow the guidelines diligently and collaborate with the security team to ensure the effective implementation of this control.

Component: Document-Oriented Database

1 Recommended countermeasures

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CM5

Very high

☐ Not tested

State:  Recommended

Description:

Conducting regular configuration audits of a Document-Oriented Database is vital for ensuring that security settings and configurations are aligned with best practices and organizational security policies. Here's how to systematically approach these audits:

1. Establish a Baseline Configuration

- Identify Security Settings:** Document the optimal security configurations for your database environment. This includes access controls, encryption settings, network configurations, and any specific database security features.
- Create a Configuration Template:** Develop a template or checklist that outlines all security-related configuration settings. This template will serve as the baseline for regular audits.

2. Automate the Audit Process

- Use Configuration Management Tools:** Implement tools that can automate the process of detecting deviations from the baseline configuration. Tools like Puppet, Chef, Ansible, or database-specific management solutions can help.
- Schedule Regular Scans:** Configure these tools to perform regular scans of your document-oriented database environment to identify and report any configuration changes or non-compliance with the baseline.

3. Review Access Controls

- Validate User Access:** Regularly review user accounts, roles, and permissions to ensure that they adhere to the principle of least privilege. Remove or adjust any excessive permissions.
- Audit Authentication Mechanisms:** Verify that authentication methods (password policies, multi-factor authentication, etc.) are properly configured and enforced.

4. Verify Network Configurations

- Check Network Access Controls:** Ensure that the database is not exposed to unauthorized networks. Validate firewall rules and other network security configurations that control access to the database.
- Secure Communication Channels:** Confirm that data in transit is encrypted and that secure communication protocols are in use.

5. Confirm Data Protection Measures

- Data Encryption:** Verify that data at rest is encrypted according to best practices. Check the configuration of encryption mechanisms and the management of encryption keys.
- Backup and Recovery:** Review the configuration of backup processes to ensure that data is securely backed up and that encryption is maintained for backups.

6. Document and Resolve Findings

- Report Findings:** Document any deviations from the baseline configuration identified during the audit. Classify findings based on their potential impact on security.
- Plan Remediation:** Develop a plan for addressing high-priority findings. This may include reconfiguring settings, updating software, or modifying access controls.
- Track Progress:** Maintain records of audits, findings, and remediation efforts to track improvements over time and to provide accountability.

7. Update Baseline Configurations

- **Incorporate Changes:** As you make changes to your database environment or as new security best practices emerge, update your baseline configuration and templates accordingly.
 - **Review Periodically:** Regularly review the baseline configuration to ensure it remains relevant and effective in mitigating security risks.
8. Educate and Train
- **Team Awareness:** Ensure that your team is aware of the configuration audit process, the importance of adhering to the baseline configurations, and the role they play in maintaining database security.
 - **Continuous Learning:** Stay informed about new security features offered by your document-oriented database system and incorporate relevant features into your security baseline.
- References and Good Security Practices**
- **Industry Best Practices:** Follow guidelines and security checklists provided by the database vendor, as well as security standards relevant to your industry (e.g., CIS Benchmarks, NIST Guidelines).
 - **Security Communities:** Engage with security communities and forums related to your database technology to stay updated on the latest security advisories and recommendations.
- Regular configuration audits are a proactive step in maintaining the security and integrity of your document-oriented database, helping to prevent unauthorized access, data leaks, and ensuring compliance with data protection regulations.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

- Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High
- Rec2. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09 Medium
- Rec3. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12 Very high

Component: API Service

- Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high

Component: Document-Oriented Database

- Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CM5 Very high

PCI-DSS-v3.2.1: 12.5.2

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

2 Recommended countermeasures

- Rec1. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09 Medium Not tested
- State: Recommended
 - Description:
To ensure comprehensive visibility into system activities, capture detailed logs for all critical actions within the web application. These logs should include data such as user actions, system changes, and access to sensitive resources. Additionally, proactively monitor these logs to detect suspicious behavior, unauthorized access, or anomalous activity, enabling timely detection of security incidents and faster response to potential threats.
- Implementation Steps:**
1. **Enable Logging for Critical Actions:** Ensure that all key actions (e.g., login attempts, data access, configuration changes) are logged in detail, including relevant metadata such as timestamps, user identifiers, and source IP addresses.
 2. **Use Centralized Logging:** Store logs in a centralized location for easier access and analysis. Integrate with a log management or SIEM system for real-time monitoring.
 3. **Set Up Automated Alerts:** Configure automated alerts for suspicious or unauthorized actions, such as failed login attempts, unexpected configuration changes, or access to sensitive data by unauthorized users.
 4. **Regularly Review Logs:** Conduct regular log reviews and audits to ensure that logging mechanisms are functioning correctly and that logs provide the necessary level of detail to identify security incidents.
- References:**
- [OWASP Logging Best Practices](#)
- Rec2. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12 Very high Not tested
- State: Recommended
 - Description:
To prevent workflow abuses, especially for critical or sensitive transactions, implement server-side checks and multi-step validation. This ensures that transactions are properly validated at multiple points in the process, reducing the risk of unauthorized actions, fraud, or exploitation of business workflows.
- Implementation Steps:**
1. **Define Critical Transactions:** Identify which transactions are critical (e.g., financial transactions, user permissions changes) and require extra validation to ensure security.
 2. **Server-Side Validation:** Ensure that all business rules, permissions, and workflows are validated on the server side, rather than relying solely on client-side validation.
 3. **Multi-Step Validation Process:** Implement a multi-step validation process for important transactions, requiring approval or confirmation at multiple stages (e.g., email confirmation, admin approval, CAPTCHA verification).
 4. **Audit Trail:** Maintain a detailed audit trail for critical transactions, logging every step and any changes made, so that any suspicious activity can be detected and reviewed.
 5. **Review and Update Validation Rules Regularly:** Periodically review and update the validation rules to adapt to new business requirements and emerging threats.
- References:**
- [PCI DSS Quick Reference Guide](#)

Component: API Service

1 Recommended countermeasures

- Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high Not tested
- State: Recommended
 - Description:
In order to fortify the security posture of our API Service, it is imperative to implement a robust logging mechanism and seamlessly integrate it with a Security Information and Event Management (SIEM) system. This control ensures that all relevant activities and events within the API ecosystem are recorded, monitored, and analyzed in real-time, enhancing our ability to detect and respond to security incidents promptly.
- Implementation Guidelines:**

Granular Logging:

- Implement detailed and granular logging mechanisms to capture pertinent information about API requests, responses, user authentication, and authorization events.
- Log relevant metadata such as timestamps, source IP addresses, user identities, and the nature of the API transactions.

Sensitive Data Masking:

- Apply appropriate data masking techniques to prevent the logging of sensitive information such as passwords, personal identifiable information (PII), and other confidential data.

Logging Standards:

- Adhere to industry-standard logging formats (e.g., JSON, syslog) to ensure compatibility with SIEM solutions and facilitate efficient log parsing and analysis.

Log Retention and Rotation:

- Define a log retention policy to ensure logs are retained for an appropriate duration based on compliance requirements and security best practices.
- Implement log rotation to prevent storage overflow and maintain a manageable log volume.

Real-time Alerting:

- Integrate the logging system with a SIEM solution to enable real-time monitoring and alerting on suspicious or anomalous activities.
- Configure alerts for specific events that may indicate security incidents, such as multiple failed login attempts or unusual patterns of API access.

Access Controls for Logs:

- Implement access controls and encryption for log files to restrict access only to authorized personnel.
- Regularly review and audit access to log files to ensure compliance and detect unauthorized access.

Integration with Incident Response:

- Ensure that the logging system is seamlessly integrated with the organization's incident response processes and tools.
- Define clear procedures for incident identification, analysis, and response based on the information gathered from the logs.

Regular Monitoring and Review:

- Periodically review and analyze logs to identify trends, potential security threats, or areas for improvement in the API service's security posture.
- Conduct regular audits of the logging and SIEM integration to validate the effectiveness of the control.

Benefits:

- Early detection of security incidents.
- Improved forensic capabilities for investigating security breaches.
- Compliance with regulatory requirements regarding data protection and incident reporting.
- Enhanced visibility into API activities for both security and operational purposes.

Note to Developers: Implementing comprehensive logging and SIEM integration is not just a security requirement but a crucial aspect of maintaining the integrity and reliability of our API Service. Follow the guidelines diligently and collaborate with the security team to ensure the effective implementation of this control.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

- Rec1. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09

Medium
- Rec2. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12

Very high

Component: API Service

- Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM

Very high

PCI-DSS-v3.2.1: 9.1.1

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

3 Recommended countermeasures

Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06

High

Not tested

State:

Recommended

Description:

To minimize the attack surface and enhance security, disable any unused services, remove default accounts, and apply security updates promptly. By doing so, you reduce the risk of exploitation through unneeded services or default credentials and ensure that the system is protected from known vulnerabilities. Additionally, use secure configurations to prevent unauthorized access and ensure that the system is optimally protected.

Implementation Steps:

- Disable Unused Services:** Identify and disable any services that are not needed for the application or system to function. This limits the number of potential attack vectors.
- Remove Default Accounts:** Remove or disable default accounts and ensure that all active accounts are assigned strong, unique credentials. Avoid using default settings that might be easily guessed by attackers.
- Apply Security Updates:** Regularly check for and apply security patches to all systems and applications. Set up an automated patch management process to ensure that critical updates are applied as soon as they are released.
- Use Secure Configurations:** Ensure that all configurations, such as database, server, and application settings, follow security best practices, including strong encryption, least privilege access, and secure communication protocols.

References:

- [OWASP Secure Configuration Guide](#)

Rec2. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09

Medium

Not tested

State:

Recommended

Description:

To ensure comprehensive visibility into system activities, capture detailed logs for all critical actions within the web application. These logs should include data such as user actions, system changes, and access to sensitive resources. Additionally, proactively monitor these logs to detect suspicious behavior, unauthorized access, or anomalous activity, enabling timely detection of security incidents and faster response to potential threats.

Implementation Steps:

- Enable Logging for Critical Actions:** Ensure that all key actions (e.g., login attempts, data access, configuration changes) are logged in detail, including relevant metadata such as timestamps, user identifiers, and source IP addresses.
- Use Centralized Logging:** Store logs in a centralized location for easier access and analysis. Integrate with a log management or SIEM system for real-time monitoring.
- Set Up Automated Alerts:** Configure automated alerts for suspicious or unauthorized actions, such as failed login attempts, unexpected configuration changes, or access to sensitive data by unauthorized users.
- Regularly Review Logs:** Conduct regular log reviews and audits to ensure that logging mechanisms are functioning correctly and that logs provide the necessary level of detail to identify security incidents.

References:

- [OWASP Logging Best Practices](#)

Report: CodeSentinel Thread Modeling

Compliance report - 2025-03-14T18:54:57.395420405Z

25 of 97

Rec3. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12 Very high Not tested

- State: Recommended
- Description:

To prevent workflow abuses, especially for critical or sensitive transactions, implement server-side checks and multi-step validation. This ensures that transactions are properly validated at multiple points in the process, reducing the risk of unauthorized actions, fraud, or exploitation of business workflows.

Implementation Steps:

 - Define Critical Transactions:** Identify which transactions are critical (e.g., financial transactions, user permissions changes) and require extra validation to ensure security.
 - Server-Side Validation:** Ensure that all business rules, permissions, and workflows are validated on the server side, rather than relying solely on client-side validation.
 - Multi-Step Validation Process:** Implement a multi-step validation process for important transactions, requiring approval or confirmation at multiple stages (e.g., email confirmation, admin approval, CAPTCHA verification).
 - Audit Trail:** Maintain a detailed audit trail for critical transactions, logging every step and any changes made, so that any suspicious activity can be detected and reviewed.
 - Review and Update Validation Rules Regularly:** Periodically review and update the validation rules to adapt to new business requirements and emerging threats.

References:

 - PCI DSS Quick Reference Guide

Component: CLI Application

1 Recommended countermeasures

Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High Not tested

- State: Recommended
- Description:

To enhance the security of your CLI application, it's essential to focus on two pivotal areas: configuration management and credential management. A strategic approach to these components not only fortifies your application against unauthorized access but also ensures the safeguarding of sensitive information. Below, we detail actionable steps developers can take to implement these practices effectively:

 - Robust Configuration Management Process:**
 - Begin by establishing a standardized process for configuring systems and applications. This includes predefined security settings that align with industry best practices and compliance requirements.
 - Regularly review and update configurations to address newly identified vulnerabilities and security threats. Utilize automated tools where possible to ensure configurations remain consistent across environments.
 - Document all configuration changes and maintain version control to enable audit trails and rollback capabilities in case of issues.
 - Regular Security Reviews:**
 - Conduct periodic security reviews of system and application configurations to identify potential security gaps or misconfigurations.
 - Engage in penetration testing and vulnerability assessments to evaluate the effectiveness of current security configurations and identify areas for improvement.
 - Secure Credential Management:**
 - Store credentials (e.g., passwords, API keys, secret tokens) using encrypted storage solutions. Ensure encryption keys are managed securely and access to them is tightly controlled.
 - Avoid hard-coding credentials in source code or configuration files. Instead, use secure vaults or environment variables for managing sensitive information.
 - Implement multi-factor authentication and strong password policies to enhance the security of user accounts and protect against unauthorized access.
 - Regularly rotate credentials and keys, especially after personnel changes or when a breach is suspected.

Component: API Service

1 Recommended countermeasures

Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high Not tested

- State: Recommended
- Description:

In order to fortify the security posture of our API Service, it is imperative to implement a robust logging mechanism and seamlessly integrate it with a Security Information and Event Management (SIEM) system. This control ensures that all relevant activities and events within the API ecosystem are recorded, monitored, and analyzed in real-time, enhancing our ability to detect and respond to security incidents promptly.

Implementation Guidelines:

Granular Logging:

 - Implement detailed and granular logging mechanisms to capture pertinent information about API requests, responses, user authentication, and authorization events.
 - Log relevant metadata such as timestamps, source IP addresses, user identities, and the nature of the API transactions.

Sensitive Data Masking:

 - Apply appropriate data masking techniques to prevent the logging of sensitive information such as passwords, personal identifiable information (PII), and other confidential data.

Logging Standards:

 - Adhere to industry-standard logging formats (e.g., JSON, syslog) to ensure compatibility with SIEM solutions and facilitate efficient log parsing and analysis.

Log Retention and Rotation:

 - Define a log retention policy to ensure logs are retained for an appropriate duration based on compliance requirements and security best practices.
 - Implement log rotation to prevent storage overflow and maintain a manageable log volume.

Real-time Alerting:

 - Integrate the logging system with a SIEM solution to enable real-time monitoring and alerting on suspicious or anomalous activities.
 - Configure alerts for specific events that may indicate security incidents, such as multiple failed login attempts or unusual patterns of API access.

Access Controls for Logs:

 - Implement access controls and encryption for log files to restrict access only to authorized personnel.
 - Regularly review and audit access to log files to ensure compliance and detect unauthorized access.

Integration with Incident Response:

 - Ensure that the logging system is seamlessly integrated with the organization's incident response processes and tools.
 - Define clear procedures for incident identification, analysis, and response based on the information gathered from the logs.

Regular Monitoring and Review:

 - Periodically review and analyze logs to identify trends, potential security threats, or areas for improvement in the API service's security posture.
 - Conduct regular audits of the logging and SIEM integration to validate the effectiveness of the control.

Benefits:

 - Early detection of security incidents.
 - Improved forensic capabilities for investigating security breaches.
 - Compliance with regulatory requirements regarding data protection and incident reporting.
 - Enhanced visibility into API activities for both security and operational purposes.

Note to Developers: Implementing comprehensive logging and SIEM integration is not just a security requirement but a crucial aspect of maintaining the integrity and reliability of our API Service. Follow the guidelines diligently and collaborate with the security team to ensure the effective implementation of this control.

Component: Document-Oriented Database

1 Recommended countermeasures

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CMS Very high Not tested

- State: Recommended
- Description:

Conducting regular configuration audits of a Document-Oriented Database is vital for ensuring that security settings and configurations are aligned with best practices and organizational security policies. Here's how to systematically approach these audits:

1. Establish a Baseline Configuration
 - **Identify Security Settings:** Document the optimal security configurations for your database environment. This includes access controls, encryption settings, network configurations, and any specific database security features.
 - **Create a Configuration Template:** Develop a template or checklist that outlines all security-related configuration settings. This template will serve as the baseline for regular audits.
 2. Automate the Audit Process
 - **Use Configuration Management Tools:** Implement tools that can automate the process of detecting deviations from the baseline configuration. Tools like Puppet, Chef, Ansible, or database-specific management solutions can help.
 - **Schedule Regular Scans:** Configure these tools to perform regular scans of your document-oriented database environment to identify and report any configuration changes or non-compliance with the baseline.
 3. Review Access Controls
 - **Validate User Access:** Regularly review user accounts, roles, and permissions to ensure that they adhere to the principle of least privilege. Remove or adjust any excessive permissions.
 - **Audit Authentication Mechanisms:** Verify that authentication methods (password policies, multi-factor authentication, etc.) are properly configured and enforced.
 4. Verify Network Configurations
 - **Check Network Access Controls:** Ensure that the database is not exposed to unauthorized networks. Validate firewall rules and other network security configurations that control access to the database.
 - **Secure Communication Channels:** Confirm that data in transit is encrypted and that secure communication protocols are in use.
 5. Confirm Data Protection Measures
 - **Data Encryption:** Verify that data at rest is encrypted according to best practices. Check the configuration of encryption mechanisms and the management of encryption keys.
 - **Backup and Recovery:** Review the configuration of backup processes to ensure that data is securely backed up and that encryption is maintained for backups.
 6. Document and Resolve Findings
 - **Report Findings:** Document any deviations from the baseline configuration identified during the audit. Classify findings based on their potential impact on security.
 - **Plan Remediation:** Develop a plan for addressing high-priority findings. This may include reconfiguring settings, updating software, or modifying access controls.
 - **Track Progress:** Maintain records of audits, findings, and remediation efforts to track improvements over time and to provide accountability.
 7. Update Baseline Configurations
 - **Incorporate Changes:** As you make changes to your database environment or as new security best practices emerge, update your baseline configuration and templates accordingly.
 - **Review Periodically:** Regularly review the baseline configuration to ensure it remains relevant and effective in mitigating security risks.
 8. Educate and Train
 - **Team Awareness:** Ensure that your team is aware of the configuration audit process, the importance of adhering to the baseline configurations, and the role they play in maintaining database security.
 - **Continuous Learning:** Stay informed about new security features offered by your document-oriented database system and incorporate relevant features into your security baseline.
- References and Good Security Practices**
- **Industry Best Practices:** Follow guidelines and security checklists provided by the database vendor, as well as security standards relevant to your industry (e.g., CIS Benchmarks, NIST Guidelines).
 - **Security Communities:** Engage with security communities and forums related to your database technology to stay updated on the latest security advisories and recommendations.
- Regular configuration audits are a proactive step in maintaining the security and integrity of your document-oriented database, helping to prevent unauthorized access, data leaks, and ensuring compliance with data protection regulations.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

- Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High
- Rec2. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09 Medium
- Rec3. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12 Very high

Component: CLI Application

- Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High

Component: API Service

- Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high

Component: Document-Oriented Database

- Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CM5 Very high

PCI-DSS-v3.2.1: 10.1

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

3 Recommended countermeasures

- Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High Not tested

State: Recommended

Description:

To minimize the attack surface and enhance security, disable any unused services, remove default accounts, and apply security updates promptly. By doing so, you reduce the risk of exploitation through unneeded services or default credentials and ensure that the system is protected from known vulnerabilities. Additionally, use secure configurations to prevent unauthorized access and ensure that the system is optimally protected.

Implementation Steps:
 1. **Disable Unused Services:** Identify and disable any services that are not needed for the application or system to function. This limits the number of potential attack vectors.
 2. **Remove Default Accounts:** Remove or disable default accounts and ensure that all active accounts are assigned strong, unique credentials. Avoid using default settings that might be easily guessed by attackers.
 3. **Apply Security Updates:** Regularly check for and apply security patches to all systems and applications. Set up an automated patch management process to ensure that critical updates are applied as soon as they are released.
 4. **Use Secure Configurations:** Ensure that all configurations, such as database, server, and application settings, follow security best practices, including strong encryption, least privilege access, and secure communication protocols.

References:

- [OWASP Secure Configuration Guide](#)

Rec2. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09 Medium Not tested

Report: CodeSentinel Thread Modeling

Compliance report - 2025-03-14T18:54:57.395420405Z

27 of 97

• State:

Recommended

• Description:

To ensure comprehensive visibility into system activities, capture detailed logs for all critical actions within the web application. These logs should include data such as user actions, system changes, and access to sensitive resources. Additionally, proactively monitor these logs to detect suspicious behavior, unauthorized access, or anomalous activity, enabling timely detection of security incidents and faster response to potential threats.

Implementation Steps:

- Enable Logging for Critical Actions:** Ensure that all key actions (e.g., login attempts, data access, configuration changes) are logged in detail, including relevant metadata such as timestamps, user identifiers, and source IP addresses.
- Use Centralized Logging:** Store logs in a centralized location for easier access and analysis. Integrate with a log management or SIEM system for real-time monitoring.
- Set Up Automated Alerts:** Configure automated alerts for suspicious or unauthorized actions, such as failed login attempts, unexpected configuration changes, or access to sensitive data by unauthorized users.
- Regularly Review Logs:** Conduct regular log reviews and audits to ensure that logging mechanisms are functioning correctly and that logs provide the necessary level of detail to identify security incidents.

References:

- [OWASP Logging Best Practices](#)

Rec3. Implement server-side checks and multi-step validation for important transactions

C-WEB-APPLICATION-SERVER-SIDE-CNT-12

Very high

Not tested

• State:

Recommended

• Description:

To prevent workflow abuses, especially for critical or sensitive transactions, implement server-side checks and multi-step validation. This ensures that transactions are properly validated at multiple points in the process, reducing the risk of unauthorized actions, fraud, or exploitation of business workflows.

Implementation Steps:

- Define Critical Transactions:** Identify which transactions are critical (e.g., financial transactions, user permissions changes) and require extra validation to ensure security.
- Server-Side Validation:** Ensure that all business rules, permissions, and workflows are validated on the server side, rather than relying solely on client-side validation.
- Multi-Step Validation Process:** Implement a multi-step validation process for important transactions, requiring approval or confirmation at multiple stages (e.g., email confirmation, admin approval, CAPTCHA verification).
- Audit Trail:** Maintain a detailed audit trail for critical transactions, logging every step and any changes made, so that any suspicious activity can be detected and reviewed.
- Review and Update Validation Rules Regularly:** Periodically review and update the validation rules to adapt to new business requirements and emerging threats.

References:

- [PCI DSS Quick Reference Guide](#)

Component: API Service

1 Recommended countermeasures

Rec1. Implement Comprehensive Logging and SIEM Integration

C-API-SERVICE-SIEM

Very high

Not tested

• State:

Recommended

• Description:

In order to fortify the security posture of our API Service, it is imperative to implement a robust logging mechanism and seamlessly integrate it with a Security Information and Event Management (SIEM) system. This control ensures that all relevant activities and events within the API ecosystem are recorded, monitored, and analyzed in real-time, enhancing our ability to detect and respond to security incidents promptly.

Implementation Guidelines:

Granular Logging:

- Implement detailed and granular logging mechanisms to capture pertinent information about API requests, responses, user authentication, and authorization events.
- Log relevant metadata such as timestamps, source IP addresses, user identities, and the nature of the API transactions.

Sensitive Data Masking:

- Apply appropriate data masking techniques to prevent the logging of sensitive information such as passwords, personal identifiable information (PII), and other confidential data.

Logging Standards:

- Adhere to industry-standard logging formats (e.g., JSON, syslog) to ensure compatibility with SIEM solutions and facilitate efficient log parsing and analysis.

Log Retention and Rotation:

- Define a log retention policy to ensure logs are retained for an appropriate duration based on compliance requirements and security best practices.
- Implement log rotation to prevent storage overflow and maintain a manageable log volume.

Real-time Alerting:

- Integrate the logging system with a SIEM solution to enable real-time monitoring and alerting on suspicious or anomalous activities.
- Configure alerts for specific events that may indicate security incidents, such as multiple failed login attempts or unusual patterns of API access.

Access Controls for Logs:

- Implement access controls and encryption for log files to restrict access only to authorized personnel.
- Regularly review and audit access to log files to ensure compliance and detect unauthorized access.

Integration with Incident Response:

- Ensure that the logging system is seamlessly integrated with the organization's incident response processes and tools.
- Define clear procedures for incident identification, analysis, and response based on the information gathered from the logs.

Regular Monitoring and Review:

- Periodically review and analyze logs to identify trends, potential security threats, or areas for improvement in the API service's security posture.
- Conduct regular audits of the logging and SIEM integration to validate the effectiveness of the control.

Benefits:

- Early detection of security incidents.
- Improved forensic capabilities for investigating security breaches.
- Compliance with regulatory requirements regarding data protection and incident reporting.
- Enhanced visibility into API activities for both security and operational purposes.

Note to Developers: Implementing comprehensive logging and SIEM integration is not just a security requirement but a crucial aspect of maintaining the integrity and reliability of our API Service. Follow the guidelines diligently and collaborate with the security team to ensure the effective implementation of this control.

Component: Document-Oriented Database

1 Recommended countermeasures

Rec1. Regular Configuration Audits

C-DOCUMENT-ORIENTED-DATABASE-CMS

Very high

Not tested

• State:

Recommended

• Description:

Conducting regular configuration audits of a Document-Oriented Database is vital for ensuring that security settings and configurations are aligned with best practices and organizational security policies. Here's how to systematically approach these audits:

- Establish a Baseline Configuration
 - Identify Security Settings:** Document the optimal security configurations for your database environment. This includes access controls, encryption settings, network configurations, and any specific database security features.
 - Create a Configuration Template:** Develop a template or checklist that outlines all security-related configuration settings. This template will serve as the baseline for regular audits.
- Automate the Audit Process
 - Use Configuration Management Tools:** Implement tools that can automate the process of detecting deviations from the baseline configuration. Tools like Puppet, Chef, Ansible, or database-specific management solutions can help.
 - Schedule Regular Scans:** Configure these tools to perform regular scans of your document-oriented database environment to identify and report any configuration changes or non-compliance with the baseline.

3. Review Access Controls
 - **Validate User Access:** Regularly review user accounts, roles, and permissions to ensure that they adhere to the principle of least privilege. Remove or adjust any excessive permissions.
 - **Audit Authentication Mechanisms:** Verify that authentication methods (password policies, multi-factor authentication, etc.) are properly configured and enforced.
4. Verify Network Configurations
 - **Check Network Access Controls:** Ensure that the database is not exposed to unauthorized networks. Validate firewall rules and other network security configurations that control access to the database.
 - **Secure Communication Channels:** Confirm that data in transit is encrypted and that secure communication protocols are in use.
5. Confirm Data Protection Measures
 - **Data Encryption:** Verify that data at rest is encrypted according to best practices. Check the configuration of encryption mechanisms and the management of encryption keys.
 - **Backup and Recovery:** Review the configuration of backup processes to ensure that data is securely backed up and that encryption is maintained for backups.
6. Document and Resolve Findings
 - **Report Findings:** Document any deviations from the baseline configuration identified during the audit. Classify findings based on their potential impact on security.
 - **Plan Remediation:** Develop a plan for addressing high-priority findings. This may include reconfiguring settings, updating software, or modifying access controls.
 - **Track Progress:** Maintain records of audits, findings, and remediation efforts to track improvements over time and to provide accountability.
7. Update Baseline Configurations
 - **Incorporate Changes:** As you make changes to your database environment or as new security best practices emerge, update your baseline configuration and templates accordingly.
 - **Review Periodically:** Regularly review the baseline configuration to ensure it remains relevant and effective in mitigating security risks.
8. Educate and Train
 - **Team Awareness:** Ensure that your team is aware of the configuration audit process, the importance of adhering to the baseline configurations, and the role they play in maintaining database security.
 - **Continuous Learning:** Stay informed about new security features offered by your document-oriented database system and incorporate relevant features into your security baseline.
- References and Good Security Practices
 - **Industry Best Practices:** Follow guidelines and security checklists provided by the database vendor, as well as security standards relevant to your industry (e.g., CIS Benchmarks, NIST Guidelines).
 - **Security Communities:** Engage with security communities and forums related to your database technology to stay updated on the latest security advisories and recommendations.

Regular configuration audits are a proactive step in maintaining the security and integrity of your document-oriented database, helping to prevent unauthorized access, data leaks, and ensuring compliance with data protection regulations.
- Recommended countermeasures
- Below are the recommended countermeasures ("Rec") by component and threat for standard reference.
- Component: Web Application - Server side
- Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High

Rec2. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09 Medium

Rec3. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12 Very high
- Component: API Service
- Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high
- Component: Document-Oriented Database
- Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CM5 Very high
- PCI-DSS-v3.2.1: 10.2
- Non-compliant countermeasures
- "Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.
- Component: Web Application - Server side
- 2 Recommended countermeasures
- Rec1. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09 Medium Not tested

State: Recommended

Description:

To ensure comprehensive visibility into system activities, capture detailed logs for all critical actions within the web application. These logs should include data such as user actions, system changes, and access to sensitive resources. Additionally, proactively monitor these logs to detect suspicious behavior, unauthorized access, or anomalous activity, enabling timely detection of security incidents and faster response to potential threats.

Implementation Steps:

1. **Enable Logging for Critical Actions:** Ensure that all key actions (e.g., login attempts, data access, configuration changes) are logged in detail, including relevant metadata such as timestamps, user identifiers, and source IP addresses.

2. **Use Centralized Logging:** Store logs in a centralized location for easier access and analysis. Integrate with a log management or SIEM system for real-time monitoring.

3. **Set Up Automated Alerts:** Configure automated alerts for suspicious or unauthorized actions, such as failed login attempts, unexpected configuration changes, or access to sensitive data by unauthorized users.

4. **Regularly Review Logs:** Conduct regular log reviews and audits to ensure that logging mechanisms are functioning correctly and that logs provide the necessary level of detail to identify security incidents.

References:
 - [OWASP Logging Best Practices](#)

Rec2. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12 Very high Not tested

State: Recommended

Description:

To prevent workflow abuses, especially for critical or sensitive transactions, implement server-side checks and multi-step validation. This ensures that transactions are properly validated at multiple points in the process, reducing the risk of unauthorized actions, fraud, or exploitation of business workflows.

Implementation Steps:

1. **Define Critical Transactions:** Identify which transactions are critical (e.g., financial transactions, user permissions changes) and require extra validation to ensure security.

2. **Server-Side Validation:** Ensure that all business rules, permissions, and workflows are validated on the server side, rather than relying solely on client-side validation.

3. **Multi-Step Validation Process:** Implement a multi-step validation process for important transactions, requiring approval or confirmation at multiple stages (e.g., email confirmation, admin approval, CAPTCHA verification).

4. **Audit Trail:** Maintain a detailed audit trail for critical transactions, logging every step and any changes made, so that any suspicious activity can be detected and reviewed.

5. **Review and Update Validation Rules Regularly:** Periodically review and update the validation rules to adapt to new business requirements and emerging threats.

References:

Report: CodeSentinel Thread Modeling

Compliance report - 2025-03-14T18:54:57.395420405Z

29 of 97

- [PCI DSS Quick Reference Guide](#)

Component: API Service

1 Recommended countermeasures

Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high ☐ Not tested

- State: 🟢 Recommended
- Description:

In order to fortify the security posture of our API Service, it is imperative to implement a robust logging mechanism and seamlessly integrate it with a Security Information and Event Management (SIEM) system. This control ensures that all relevant activities and events within the API ecosystem are recorded, monitored, and analyzed in real-time, enhancing our ability to detect and respond to security incidents promptly.

Implementation Guidelines:

Granular Logging:

 - Implement detailed and granular logging mechanisms to capture pertinent information about API requests, responses, user authentication, and authorization events.
 - Log relevant metadata such as timestamps, source IP addresses, user identities, and the nature of the API transactions.

Sensitive Data Masking:

 - Apply appropriate data masking techniques to prevent the logging of sensitive information such as passwords, personal identifiable information (PII), and other confidential data.

Logging Standards:

 - Adhere to industry-standard logging formats (e.g., JSON, syslog) to ensure compatibility with SIEM solutions and facilitate efficient log parsing and analysis.

Log Retention and Rotation:

 - Define a log retention policy to ensure logs are retained for an appropriate duration based on compliance requirements and security best practices.
 - Implement log rotation to prevent storage overflow and maintain a manageable log volume.

Real-time Alerting:

 - Integrate the logging system with a SIEM solution to enable real-time monitoring and alerting on suspicious or anomalous activities.
 - Configure alerts for specific events that may indicate security incidents, such as multiple failed login attempts or unusual patterns of API access.

Access Controls for Logs:

 - Implement access controls and encryption for log files to restrict access only to authorized personnel.
 - Regularly review and audit access to log files to ensure compliance and detect unauthorized access.

Integration with Incident Response:

 - Ensure that the logging system is seamlessly integrated with the organization's incident response processes and tools.
 - Define clear procedures for incident identification, analysis, and response based on the information gathered from the logs.

Regular Monitoring and Review:

 - Periodically review and analyze logs to identify trends, potential security threats, or areas for improvement in the API service's security posture.
 - Conduct regular audits of the logging and SIEM integration to validate the effectiveness of the control.

Benefits:

 - Early detection of security incidents.
 - Improved forensic capabilities for investigating security breaches.
 - Compliance with regulatory requirements regarding data protection and incident reporting.
 - Enhanced visibility into API activities for both security and operational purposes.

Note to Developers: Implementing comprehensive logging and SIEM integration is not just a security requirement but a crucial aspect of maintaining the integrity and reliability of our API Service. Follow the guidelines diligently and collaborate with the security team to ensure the effective implementation of this control.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

- Rec1. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09 🟡 Medium
- Rec2. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12 🔴 Very high

Component: API Service

Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM 🔴 Very high

PCI-DSS-v3.2.1: 10.3

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

2 Recommended countermeasures

Rec1. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09 🟡 Medium ☐ Not tested

- State: 🟢 Recommended
 - Description:

To ensure comprehensive visibility into system activities, capture detailed logs for all critical actions within the web application. These logs should include data such as user actions, system changes, and access to sensitive resources. Additionally, proactively monitor these logs to detect suspicious behavior, unauthorized access, or anomalous activity, enabling timely detection of security incidents and faster response to potential threats.

Implementation Steps:

 1. **Enable Logging for Critical Actions:** Ensure that all key actions (e.g., login attempts, data access, configuration changes) are logged in detail, including relevant metadata such as timestamps, user identifiers, and source IP addresses.
 2. **Use Centralized Logging:** Store logs in a centralized location for easier access and analysis. Integrate with a log management or SIEM system for real-time monitoring.
 3. **Set Up Automated Alerts:** Configure automated alerts for suspicious or unauthorized actions, such as failed login attempts, unexpected configuration changes, or access to sensitive data by unauthorized users.
 4. **Regularly Review Logs:** Conduct regular log reviews and audits to ensure that logging mechanisms are functioning correctly and that logs provide the necessary level of detail to identify security incidents.
- References:**
- [OWASP Logging Best Practices](#)

Rec2. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12 🔴 Very high ☐ Not tested

- State: 🟢 Recommended
- Description:

To prevent workflow abuses, especially for critical or sensitive transactions, implement server-side checks and multi-step validation. This ensures that transactions are properly validated at multiple points in the process, reducing the risk of unauthorized actions, fraud, or exploitation of business workflows.

Implementation Steps:

- 1. **Define Critical Transactions:** Identify which transactions are critical (e.g., financial transactions, user permissions changes) and require extra validation to ensure security.
- 2. **Server-Side Validation:** Ensure that all business rules, permissions, and workflows are validated on the server side, rather than relying solely on client-side validation.
- 3. **Multi-Step Validation Process:** Implement a multi-step validation process for important transactions, requiring approval or confirmation at multiple stages (e.g., email confirmation, admin approval, CAPTCHA verification).
- 4. **Audit Trail:** Maintain a detailed audit trail for critical transactions, logging every step and any changes made, so that any suspicious activity can be detected and reviewed.
- 5. **Review and Update Validation Rules Regularly:** Periodically review and update the validation rules to adapt to new business requirements and emerging threats.

References:

- [PCI DSS Quick Reference Guide](#)

Component: API Service

1 Recommended countermeasures

Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high Not tested

- State: Recommended
- Description:

In order to fortify the security posture of our API Service, it is imperative to implement a robust logging mechanism and seamlessly integrate it with a Security Information and Event Management (SIEM) system. This control ensures that all relevant activities and events within the API ecosystem are recorded, monitored, and analyzed in real-time, enhancing our ability to detect and respond to security incidents promptly.

Implementation Guidelines:

Granular Logging:

 - Implement detailed and granular logging mechanisms to capture pertinent information about API requests, responses, user authentication, and authorization events.
 - Log relevant metadata such as timestamps, source IP addresses, user identities, and the nature of the API transactions.

Sensitive Data Masking:

 - Apply appropriate data masking techniques to prevent the logging of sensitive information such as passwords, personal identifiable information (PII), and other confidential data.

Logging Standards:

 - Adhere to industry-standard logging formats (e.g., JSON, syslog) to ensure compatibility with SIEM solutions and facilitate efficient log parsing and analysis.

Log Retention and Rotation:

 - Define a log retention policy to ensure logs are retained for an appropriate duration based on compliance requirements and security best practices.
 - Implement log rotation to prevent storage overflow and maintain a manageable log volume.

Real-time Alerting:

 - Integrate the logging system with a SIEM solution to enable real-time monitoring and alerting on suspicious or anomalous activities.
 - Configure alerts for specific events that may indicate security incidents, such as multiple failed login attempts or unusual patterns of API access.

Access Controls for Logs:

 - Implement access controls and encryption for log files to restrict access only to authorized personnel.
 - Regularly review and audit access to log files to ensure compliance and detect unauthorized access.

Integration with Incident Response:

 - Ensure that the logging system is seamlessly integrated with the organization's incident response processes and tools.
 - Define clear procedures for incident identification, analysis, and response based on the information gathered from the logs.

Regular Monitoring and Review:

 - Periodically review and analyze logs to identify trends, potential security threats, or areas for improvement in the API service's security posture.
 - Conduct regular audits of the logging and SIEM integration to validate the effectiveness of the control.

Benefits:

 - Early detection of security incidents.
 - Improved forensic capabilities for investigating security breaches.
 - Compliance with regulatory requirements regarding data protection and incident reporting.
 - Enhanced visibility into API activities for both security and operational purposes.

Note to Developers: Implementing comprehensive logging and SIEM integration is not just a security requirement but a crucial aspect of maintaining the integrity and reliability of our API Service. Follow the guidelines diligently and collaborate with the security team to ensure the effective implementation of this control.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

- Rec1. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09 Medium
- Rec2. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12 Very high

Component: API Service

- Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high

PCI-DSS-v3.2.1: 10.4

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

2 Recommended countermeasures

Rec1. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09 Medium Not tested

- State: Recommended
- Description:

To ensure comprehensive visibility into system activities, capture detailed logs for all critical actions within the web application. These logs should include data such as user actions, system changes, and access to sensitive resources. Additionally, proactively monitor these logs to detect suspicious behavior, unauthorized access, or anomalous activity, enabling timely detection of security incidents and faster response to potential threats.

Implementation Steps:

 1. **Enable Logging for Critical Actions:** Ensure that all key actions (e.g., login attempts, data access, configuration changes) are logged in detail, including relevant metadata such as timestamps, user identifiers, and source IP addresses.
 2. **Use Centralized Logging:** Store logs in a centralized location for easier access and analysis. Integrate with a log management or SIEM system for real-time monitoring.

3. **Set Up Automated Alerts:** Configure automated alerts for suspicious or unauthorized actions, such as failed login attempts, unexpected configuration changes, or access to sensitive data by unauthorized users.
4. **Regularly Review Logs:** Conduct regular log reviews and audits to ensure that logging mechanisms are functioning correctly and that logs provide the necessary level of detail to identify security incidents.

- References:
- OWASP Logging Best Practices

Rec2. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12 Very high Not tested

- State: Recommended
 - Description:

To prevent workflow abuses, especially for critical or sensitive transactions, implement server-side checks and multi-step validation. This ensures that transactions are properly validated at multiple points in the process, reducing the risk of unauthorized actions, fraud, or exploitation of business workflows.

Implementation Steps:

 - Define Critical Transactions:** Identify which transactions are critical (e.g., financial transactions, user permissions changes) and require extra validation to ensure security.
 - Server-Side Validation:** Ensure that all business rules, permissions, and workflows are validated on the server side, rather than relying solely on client-side validation.
 - Multi-Step Validation Process:** Implement a multi-step validation process for important transactions, requiring approval or confirmation at multiple stages (e.g., email confirmation, admin approval, CAPTCHA verification).
 - Audit Trail:** Maintain a detailed audit trail for critical transactions, logging every step and any changes made, so that any suspicious activity can be detected and reviewed.
 - Review and Update Validation Rules Regularly:** Periodically review and update the validation rules to adapt to new business requirements and emerging threats.
- References:
- PCI DSS Quick Reference Guide

Component: API Service

1 Recommended countermeasures

Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high Not tested

- State: Recommended
- Description:

In order to fortify the security posture of our API Service, it is imperative to implement a robust logging mechanism and seamlessly integrate it with a Security Information and Event Management (SIEM) system. This control ensures that all relevant activities and events within the API ecosystem are recorded, monitored, and analyzed in real-time, enhancing our ability to detect and respond to security incidents promptly.

Implementation Guidelines:

Granular Logging:

 - Implement detailed and granular logging mechanisms to capture pertinent information about API requests, responses, user authentication, and authorization events.
 - Log relevant metadata such as timestamps, source IP addresses, user identities, and the nature of the API transactions.

Sensitive Data Masking:

 - Apply appropriate data masking techniques to prevent the logging of sensitive information such as passwords, personal identifiable information (PII), and other confidential data.

Logging Standards:

 - Adhere to industry-standard logging formats (e.g., JSON, syslog) to ensure compatibility with SIEM solutions and facilitate efficient log parsing and analysis.

Log Retention and Rotation:

 - Define a log retention policy to ensure logs are retained for an appropriate duration based on compliance requirements and security best practices.
 - Implement log rotation to prevent storage overflow and maintain a manageable log volume.

Real-time Alerting:

 - Integrate the logging system with a SIEM solution to enable real-time monitoring and alerting on suspicious or anomalous activities.
 - Configure alerts for specific events that may indicate security incidents, such as multiple failed login attempts or unusual patterns of API access.

Access Controls for Logs:

 - Implement access controls and encryption for log files to restrict access only to authorized personnel.
 - Regularly review and audit access to log files to ensure compliance and detect unauthorized access.

Integration with Incident Response:

 - Ensure that the logging system is seamlessly integrated with the organization's incident response processes and tools.
 - Define clear procedures for incident identification, analysis, and response based on the information gathered from the logs.

Regular Monitoring and Review:

 - Periodically review and analyze logs to identify trends, potential security threats, or areas for improvement in the API service's security posture.
 - Conduct regular audits of the logging and SIEM integration to validate the effectiveness of the control.

Benefits:

 - Early detection of security incidents.
 - Improved forensic capabilities for investigating security breaches.
 - Compliance with regulatory requirements regarding data protection and incident reporting.
 - Enhanced visibility into API activities for both security and operational purposes.

Note to Developers: Implementing comprehensive logging and SIEM integration is not just a security requirement but a crucial aspect of maintaining the integrity and reliability of our API Service. Follow the guidelines diligently and collaborate with the security team to ensure the effective implementation of this control.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

- Rec1. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09 Medium
- Rec2. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12 Very high

Component: API Service

- Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high

PCI-DSS-v3.2.1: 10.5

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

2 Recommended countermeasures

Rec1. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09 Medium ☐ Not tested

- State: Recommended
- Description:

To ensure comprehensive visibility into system activities, capture detailed logs for all critical actions within the web application. These logs should include data such as user actions, system changes, and access to sensitive resources. Additionally, proactively monitor these logs to detect suspicious behavior, unauthorized access, or anomalous activity, enabling timely detection of security incidents and faster response to potential threats.

Implementation Steps:

 - Enable Logging for Critical Actions:** Ensure that all key actions (e.g., login attempts, data access, configuration changes) are logged in detail, including relevant metadata such as timestamps, user identifiers, and source IP addresses.
 - Use Centralized Logging:** Store logs in a centralized location for easier access and analysis. Integrate with a log management or SIEM system for real-time monitoring.
 - Set Up Automated Alerts:** Configure automated alerts for suspicious or unauthorized actions, such as failed login attempts, unexpected configuration changes, or access to sensitive data by unauthorized users.
 - Regularly Review Logs:** Conduct regular log reviews and audits to ensure that logging mechanisms are functioning correctly and that logs provide the necessary level of detail to identify security incidents.

References:

 - [OWASP Logging Best Practices](#)

Rec2. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12 Very high ☐ Not tested

- State: Recommended
- Description:

To prevent workflow abuses, especially for critical or sensitive transactions, implement server-side checks and multi-step validation. This ensures that transactions are properly validated at multiple points in the process, reducing the risk of unauthorized actions, fraud, or exploitation of business workflows.

Implementation Steps:

 - Define Critical Transactions:** Identify which transactions are critical (e.g., financial transactions, user permissions changes) and require extra validation to ensure security.
 - Server-Side Validation:** Ensure that all business rules, permissions, and workflows are validated on the server side, rather than relying solely on client-side validation.
 - Multi-Step Validation Process:** Implement a multi-step validation process for important transactions, requiring approval or confirmation at multiple stages (e.g., email confirmation, admin approval, CAPTCHA verification).
 - Audit Trail:** Maintain a detailed audit trail for critical transactions, logging every step and any changes made, so that any suspicious activity can be detected and reviewed.
 - Review and Update Validation Rules Regularly:** Periodically review and update the validation rules to adapt to new business requirements and emerging threats.

References:

 - [PCI DSS Quick Reference Guide](#)

Component: API Service

1 Recommended countermeasures

Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high ☐ Not tested

- State: Recommended
- Description:

In order to fortify the security posture of our API Service, it is imperative to implement a robust logging mechanism and seamlessly integrate it with a Security Information and Event Management (SIEM) system. This control ensures that all relevant activities and events within the API ecosystem are recorded, monitored, and analyzed in real-time, enhancing our ability to detect and respond to security incidents promptly.

Implementation Guidelines:

Granular Logging:

 - Implement detailed and granular logging mechanisms to capture pertinent information about API requests, responses, user authentication, and authorization events.
 - Log relevant metadata such as timestamps, source IP addresses, user identities, and the nature of the API transactions.

Sensitive Data Masking:

 - Apply appropriate data masking techniques to prevent the logging of sensitive information such as passwords, personal identifiable information (PII), and other confidential data.

Logging Standards:

 - Adhere to industry-standard logging formats (e.g., JSON, syslog) to ensure compatibility with SIEM solutions and facilitate efficient log parsing and analysis.

Log Retention and Rotation:

 - Define a log retention policy to ensure logs are retained for an appropriate duration based on compliance requirements and security best practices.
 - Implement log rotation to prevent storage overflow and maintain a manageable log volume.

Real-time Alerting:

 - Integrate the logging system with a SIEM solution to enable real-time monitoring and alerting on suspicious or anomalous activities.
 - Configure alerts for specific events that may indicate security incidents, such as multiple failed login attempts or unusual patterns of API access.

Access Controls for Logs:

 - Implement access controls and encryption for log files to restrict access only to authorized personnel.
 - Regularly review and audit access to log files to ensure compliance and detect unauthorized access.

Integration with Incident Response:

 - Ensure that the logging system is seamlessly integrated with the organization's incident response processes and tools.
 - Define clear procedures for incident identification, analysis, and response based on the information gathered from the logs.

Regular Monitoring and Review:

 - Periodically review and analyze logs to identify trends, potential security threats, or areas for improvement in the API service's security posture.
 - Conduct regular audits of the logging and SIEM integration to validate the effectiveness of the control.

Benefits:

 - Early detection of security incidents.
 - Improved forensic capabilities for investigating security breaches.
 - Compliance with regulatory requirements regarding data protection and incident reporting.
 - Enhanced visibility into API activities for both security and operational purposes.

Note to Developers: Implementing comprehensive logging and SIEM integration is not just a security requirement but a crucial aspect of maintaining the integrity and reliability of our API Service. Follow the guidelines diligently and collaborate with the security team to ensure the effective implementation of this control.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

- Rec1. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09 Medium
- Rec2. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12 Very high

Component: API Service

- Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high

PCI-DSS-v3.2.1: 10.6

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

4 Recommended countermeasures

Rec1. Check user roles and permissions on every request

C-WEB-APPLICATION-SERVER-SIDE-CNT-03

High

Not tested

State: Recommended

Description:

To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.

Implementation Steps:

- Role-Based Access Control (RBAC):** Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.
- Permission Validation:** For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.
- Enforce Fine-Grained Access Control:** Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.
- Session Management:** Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.
- Audit Logs:** Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.

References:

[OWASP Access Control Cheat Sheet](#)

Rec2. Block or validate all outbound requests

C-WEB-APPLICATION-SERVER-SIDE-CNT-11

Medium

Not tested

State: Recommended

Description:

To reduce the risk of data exfiltration or unauthorized system communication, block or carefully validate all outbound requests made by the application. Ensure that only requests to specific, trusted domains and IP addresses are allowed, and that any outbound request to unapproved destinations is blocked or flagged for further review. This control prevents the application from communicating with malicious external systems and protects sensitive data from being leaked.

Implementation Steps:

- Implement Outbound Request Filtering:** Use a web application firewall (WAF) or proxy to block or monitor any outbound requests that are not to trusted domains or IP addresses.
- Validate Outbound Requests:** Before allowing outbound communication, validate the destination domain and IP address to ensure they are within an approved list of resources necessary for the application.
- Set Domain and IP Allowlists:** Define an allowlist of specific domains and IP addresses that the application is authorized to interact with, and block all other outbound traffic.
- Monitor and Log Outbound Requests:** Continuously monitor and log all outbound requests for suspicious activity or attempts to communicate with unapproved destinations.
- Review and Update Allowlist Regularly:** Periodically review the allowlist to ensure that it remains up-to-date with the application's legitimate requirements.

References:

[Security and Privacy Controls for Information Systems and Organizations](#)

Rec3. Capture and monitor detailed logs for critical actions

C-WEB-APPLICATION-SERVER-SIDE-CNT-09

Medium

Not tested

State: Recommended

Description:

To ensure comprehensive visibility into system activities, capture detailed logs for all critical actions within the web application. These logs should include data such as user actions, system changes, and access to sensitive resources. Additionally, proactively monitor these logs to detect suspicious behavior, unauthorized access, or anomalous activity, enabling timely detection of security incidents and faster response to potential threats.

Implementation Steps:

- Enable Logging for Critical Actions:** Ensure that all key actions (e.g., login attempts, data access, configuration changes) are logged in detail, including relevant metadata such as timestamps, user identifiers, and source IP addresses.
- Use Centralized Logging:** Store logs in a centralized location for easier access and analysis. Integrate with a log management or SIEM system for real-time monitoring.
- Set Up Automated Alerts:** Configure automated alerts for suspicious or unauthorized actions, such as failed login attempts, unexpected configuration changes, or access to sensitive data by unauthorized users.
- Regularly Review Logs:** Conduct regular log reviews and audits to ensure that logging mechanisms are functioning correctly and that logs provide the necessary level of detail to identify security incidents.

References:

[OWASP Logging Best Practices](#)

Rec4. Implement server-side checks and multi-step validation for important transactions

C-WEB-APPLICATION-SERVER-SIDE-CNT-12

Very high

Not tested

State: Recommended

Description:

To prevent workflow abuses, especially for critical or sensitive transactions, implement server-side checks and multi-step validation. This ensures that transactions are properly validated at multiple points in the process, reducing the risk of unauthorized actions, fraud, or exploitation of business workflows.

Implementation Steps:

- Define Critical Transactions:** Identify which transactions are critical (e.g., financial transactions, user permissions changes) and require extra validation to ensure security.
- Server-Side Validation:** Ensure that all business rules, permissions, and workflows are validated on the server side, rather than relying solely on client-side validation.
- Multi-Step Validation Process:** Implement a multi-step validation process for important transactions, requiring approval or confirmation at multiple stages (e.g., email confirmation, admin approval, CAPTCHA verification).
- Audit Trail:** Maintain a detailed audit trail for critical transactions, logging every step and any changes made, so that any suspicious activity can be detected and reviewed.
- Review and Update Validation Rules Regularly:** Periodically review and update the validation rules to adapt to new business requirements and emerging threats.

References:

[PCI DSS Quick Reference Guide](#)

Component: CLI Application

1 Recommended countermeasures

Rec1. Comprehensive security configuration and credential management

C-CLI-APPLICATION-C003

High

Not tested

State: Recommended

Description:

To enhance the security of your CLI application, it's essential to focus on two pivotal areas: configuration management and credential management. A strategic approach to these components not only fortifies your application against unauthorized access but also ensures the safeguarding of sensitive information. Below, we detail actionable steps developers can take to implement these practices effectively:

Report: CodeSentinel Thread Modeling

Compliance report - 2025-03-14T18:54:57.395420405Z


34 of 97

- **Robust Configuration Management Process:**
 - Begin by establishing a standardized process for configuring systems and applications. This includes predefined security settings that align with industry best practices and compliance requirements.
 - Regularly review and update configurations to address newly identified vulnerabilities and security threats. Utilize automated tools where possible to ensure configurations remain consistent across environments.
 - Document all configuration changes and maintain version control to enable audit trails and rollback capabilities in case of issues.
- **Regular Security Reviews:**
 - Conduct periodic security reviews of system and application configurations to identify potential security gaps or misconfigurations.
 - Engage in penetration testing and vulnerability assessments to evaluate the effectiveness of current security configurations and identify areas for improvement.
- **Secure Credential Management:**
 - Store credentials (e.g., passwords, API keys, secret tokens) using encrypted storage solutions. Ensure encryption keys are managed securely and access to them is tightly controlled.
 - Avoid hard-coding credentials in source code or configuration files. Instead, use secure vaults or environment variables for managing sensitive information.
 - Implement multi-factor authentication and strong password policies to enhance the security of user accounts and protect against unauthorized access.
 - Regularly rotate credentials and keys, especially after personnel changes or when a breach is suspected.

Component: API Service

1 Recommended countermeasures

Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high ☐ Not tested

- State:  Recommended
- Description:

In order to fortify the security posture of our API Service, it is imperative to implement a robust logging mechanism and seamlessly integrate it with a Security Information and Event Management (SIEM) system. This control ensures that all relevant activities and events within the API ecosystem are recorded, monitored, and analyzed in real-time, enhancing our ability to detect and respond to security incidents promptly.

Implementation Guidelines:

Granular Logging:

 - Implement detailed and granular logging mechanisms to capture pertinent information about API requests, responses, user authentication, and authorization events.
 - Log relevant metadata such as timestamps, source IP addresses, user identities, and the nature of the API transactions.

Sensitive Data Masking:

 - Apply appropriate data masking techniques to prevent the logging of sensitive information such as passwords, personal identifiable information (PII), and other confidential data.

Logging Standards:

 - Adhere to industry-standard logging formats (e.g., JSON, syslog) to ensure compatibility with SIEM solutions and facilitate efficient log parsing and analysis.

Log Retention and Rotation:

 - Define a log retention policy to ensure logs are retained for an appropriate duration based on compliance requirements and security best practices.
 - Implement log rotation to prevent storage overflow and maintain a manageable log volume.

Real-time Alerting:

 - Integrate the logging system with a SIEM solution to enable real-time monitoring and alerting on suspicious or anomalous activities.
 - Configure alerts for specific events that may indicate security incidents, such as multiple failed login attempts or unusual patterns of API access.

Access Controls for Logs:

 - Implement access controls and encryption for log files to restrict access only to authorized personnel.
 - Regularly review and audit access to log files to ensure compliance and detect unauthorized access.

Integration with Incident Response:

 - Ensure that the logging system is seamlessly integrated with the organization's incident response processes and tools.
 - Define clear procedures for incident identification, analysis, and response based on the information gathered from the logs.

Regular Monitoring and Review:

 - Periodically review and analyze logs to identify trends, potential security threats, or areas for improvement in the API service's security posture.
 - Conduct regular audits of the logging and SIEM integration to validate the effectiveness of the control.

Benefits:


 - Early detection of security incidents.
 - Improved forensic capabilities for investigating security breaches.
 - Compliance with regulatory requirements regarding data protection and incident reporting.
 - Enhanced visibility into API activities for both security and operational purposes.

Note to Developers: Implementing comprehensive logging and SIEM integration is not just a security requirement but a crucial aspect of maintaining the integrity and reliability of our API Service. Follow the guidelines diligently and collaborate with the security team to ensure the effective implementation of this control.

Component: Source Code Repository

1 Recommended countermeasures

Rec1. Enforce a model of least privilege for source code repositories C-SOURCE-CODE-REPOSITORY-CNT-03  High ☐ Not tested

- State:  Recommended
- Description:

Least privilege is the principle that a security architecture should be designed so that each entity is granted the minimum system resources and authorizations that the entity needs to perform its function.

Remediation:
Consider the exposure of your repository. Technically enforce a model of least privilege for who can make changes to your code repository. Make sure all activity is attributable. Additionally, access to the repository should be revoked swiftly when no longer required, or in the event of compromise

Reference:

 - [Protect access credentials](#)

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

- Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03  High
- Rec2. Block or validate all outbound requests C-WEB-APPLICATION-SERVER-SIDE-CNT-11  Medium
- Rec3. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09  Medium
- Rec4. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12 Very high

Component: CLI Application

- Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003  High

Component: API Service

- Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high

Component: Source Code Repository

Rec1. Enforce a model of least privilege for source code repositories C-SOURCE-CODE-REPOSITORY-CNT-03 High

PCI-DSS-v3.2.1: 10.7

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

2 Recommended countermeasures

Rec1. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09 Medium Not tested

- State: Recommended
- Description:

To ensure comprehensive visibility into system activities, capture detailed logs for all critical actions within the web application. These logs should include data such as user actions, system changes, and access to sensitive resources. Additionally, proactively monitor these logs to detect suspicious behavior, unauthorized access, or anomalous activity, enabling timely detection of security incidents and faster response to potential threats.

Implementation Steps:

 - Enable Logging for Critical Actions:** Ensure that all key actions (e.g., login attempts, data access, configuration changes) are logged in detail, including relevant metadata such as timestamps, user identifiers, and source IP addresses.
 - Use Centralized Logging:** Store logs in a centralized location for easier access and analysis. Integrate with a log management or SIEM system for real-time monitoring.
 - Set Up Automated Alerts:** Configure automated alerts for suspicious or unauthorized actions, such as failed login attempts, unexpected configuration changes, or access to sensitive data by unauthorized users.
 - Regularly Review Logs:** Conduct regular log reviews and audits to ensure that logging mechanisms are functioning correctly and that logs provide the necessary level of detail to identify security incidents.

References:

 - [OWASP Logging Best Practices](#)

Rec2. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12 Very high Not tested

- State: Recommended
- Description:

To prevent workflow abuses, especially for critical or sensitive transactions, implement server-side checks and multi-step validation. This ensures that transactions are properly validated at multiple points in the process, reducing the risk of unauthorized actions, fraud, or exploitation of business workflows.

Implementation Steps:

 - Define Critical Transactions:** Identify which transactions are critical (e.g., financial transactions, user permissions changes) and require extra validation to ensure security.
 - Server-Side Validation:** Ensure that all business rules, permissions, and workflows are validated on the server side, rather than relying solely on client-side validation.
 - Multi-Step Validation Process:** Implement a multi-step validation process for important transactions, requiring approval or confirmation at multiple stages (e.g., email confirmation, admin approval, CAPTCHA verification).
 - Audit Trail:** Maintain a detailed audit trail for critical transactions, logging every step and any changes made, so that any suspicious activity can be detected and reviewed.
 - Review and Update Validation Rules Regularly:** Periodically review and update the validation rules to adapt to new business requirements and emerging threats.

References:

 - [PCI DSS Quick Reference Guide](#)

Component: API Service

1 Recommended countermeasures

Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high Not tested

- State: Recommended
- Description:

In order to fortify the security posture of our API Service, it is imperative to implement a robust logging mechanism and seamlessly integrate it with a Security Information and Event Management (SIEM) system. This control ensures that all relevant activities and events within the API ecosystem are recorded, monitored, and analyzed in real-time, enhancing our ability to detect and respond to security incidents promptly.

Implementation Guidelines:

Granular Logging:

 - Implement detailed and granular logging mechanisms to capture pertinent information about API requests, responses, user authentication, and authorization events.
 - Log relevant metadata such as timestamps, source IP addresses, user identities, and the nature of the API transactions.

Sensitive Data Masking:

 - Apply appropriate data masking techniques to prevent the logging of sensitive information such as passwords, personal identifiable information (PII), and other confidential data.

Logging Standards:

 - Adhere to industry-standard logging formats (e.g., JSON, syslog) to ensure compatibility with SIEM solutions and facilitate efficient log parsing and analysis.

Log Retention and Rotation:

 - Define a log retention policy to ensure logs are retained for an appropriate duration based on compliance requirements and security best practices.
 - Implement log rotation to prevent storage overflow and maintain a manageable log volume.

Real-time Alerting:

 - Integrate the logging system with a SIEM solution to enable real-time monitoring and alerting on suspicious or anomalous activities.
 - Configure alerts for specific events that may indicate security incidents, such as multiple failed login attempts or unusual patterns of API access.

Access Controls for Logs:

 - Implement access controls and encryption for log files to restrict access only to authorized personnel.
 - Regularly review and audit access to log files to ensure compliance and detect unauthorized access.

Integration with Incident Response:

 - Ensure that the logging system is seamlessly integrated with the organization's incident response processes and tools.
 - Define clear procedures for incident identification, analysis, and response based on the information gathered from the logs.

Regular Monitoring and Review:

 - Periodically review and analyze logs to identify trends, potential security threats, or areas for improvement in the API service's security posture.
 - Conduct regular audits of the logging and SIEM integration to validate the effectiveness of the control.

Benefits:

 - Early detection of security incidents.
 - Improved forensic capabilities for investigating security breaches.
 - Compliance with regulatory requirements regarding data protection and incident reporting.
 - Enhanced visibility into API activities for both security and operational purposes.

Note to Developers: Implementing comprehensive logging and SIEM integration is not just a security requirement but a crucial aspect of maintaining the integrity and reliability of our API Service. Follow the guidelines diligently and collaborate with the security team to ensure the effective implementation of this control.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

- Rec1. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09

Medium
- Rec2. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12

Very high

Component: API Service

- Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM

Very high

PCI-DSS-v3.2.1: 10.8

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

4 Recommended countermeasures

- Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03

High

Not tested

State:

Recommended

Description:

To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.

Implementation Steps:

1. Role-Based Access Control (RBAC):

Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.

2. Permission Validation:

For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.

3. Enforce Fine-Grained Access Control:

Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.

4. Session Management:

Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.

5. Audit Logs:

Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.

References:

OWASP Access Control Cheat Sheet

Rec2. Block or validate all outbound requests C-WEB-APPLICATION-SERVER-SIDE-CNT-11

Medium

Not tested

State:

Recommended

Description:

To reduce the risk of data exfiltration or unauthorized system communication, block or carefully validate all outbound requests made by the application. Ensure that only requests to specific, trusted domains and IP addresses are allowed, and that any outbound request to unapproved destinations is blocked or flagged for further review. This control prevents the application from communicating with malicious external systems and protects sensitive data from being leaked.

Implementation Steps:

1. Implement Outbound Request Filtering:

Use a web application firewall (WAF) or proxy to block or monitor any outbound requests that are not to trusted domains or IP addresses.

2. Validate Outbound Requests:

Before allowing outbound communication, validate the destination domain and IP address to ensure they are within an approved list of resources necessary for the application.

3. Set Domain and IP Allowlists:

Define an allowlist of specific domains and IP addresses that the application is authorized to interact with, and block all other outbound traffic.

4. Monitor and Log Outbound Requests:

Continuously monitor and log all outbound requests for suspicious activity or attempts to communicate with unapproved destinations.

5. Review and Update Allowlist Regularly:

Periodically review the allowlist to ensure that it remains up-to-date with the application's legitimate requirements.

References:

Security and Privacy Controls for Information Systems and Organizations

Rec3. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09

Medium

Not tested

State:

Recommended

Description:

To ensure comprehensive visibility into system activities, capture detailed logs for all critical actions within the web application. These logs should include data such as user actions, system changes, and access to sensitive resources. Additionally, proactively monitor these logs to detect suspicious behavior, unauthorized access, or anomalous activity, enabling timely detection of security incidents and faster response to potential threats.

Implementation Steps:

1. Enable Logging for Critical Actions:

Ensure that all key actions (e.g., login attempts, data access, configuration changes) are logged in detail, including relevant metadata such as timestamps, user identifiers, and source IP addresses.

2. Use Centralized Logging:

Store logs in a centralized location for easier access and analysis. Integrate with a log management or SIEM system for real-time monitoring.

3. Set Up Automated Alerts:

Configure automated alerts for suspicious or unauthorized actions, such as failed login attempts, unexpected configuration changes, or access to sensitive data by unauthorized users.

4. Regularly Review Logs:

Conduct regular log reviews and audits to ensure that logging mechanisms are functioning correctly and that logs provide the necessary level of detail to identify security incidents.

References:

OWASP Logging Best Practices

Rec4. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12

Very high

Not tested

State:

Recommended

Description:

To prevent workflow abuses, especially for critical or sensitive transactions, implement server-side checks and multi-step validation. This ensures that transactions are properly validated at multiple points in the process, reducing the risk of unauthorized actions, fraud, or exploitation of business workflows.

Implementation Steps:

1. Define Critical Transactions:

Identify which transactions are critical (e.g., financial transactions, user permissions changes) and require extra validation to ensure security.

2. Server-Side Validation:

Ensure that all business rules, permissions, and workflows are validated on the server side, rather than relying solely on client-side validation.

3. Multi-Step Validation Process:

Implement a multi-step validation process for important transactions, requiring approval or confirmation at multiple stages (e.g., email confirmation, admin approval, CAPTCHA verification).

4. Audit Trail:

Maintain a detailed audit trail for critical transactions, logging every step and any changes made, so that any suspicious activity can be detected and reviewed.

Report: CodeSentinel Thread Modeling

Compliance report - 2025-03-14T18:54:57.395420405Z

37 of 97

5. **Review and Update Validation Rules Regularly:** Periodically review and update the validation rules to adapt to new business requirements and emerging threats.


References:

- [PCI DSS Quick Reference Guide](#)

Component: API Service

1 Recommended countermeasures

Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high ☐ Not tested

- State:  Recommended
- Description:
In order to fortify the security posture of our API Service, it is imperative to implement a robust logging mechanism and seamlessly integrate it with a Security Information and Event Management (SIEM) system. This control ensures that all relevant activities and events within the API ecosystem are recorded, monitored, and analyzed in real-time, enhancing our ability to detect and respond to security incidents promptly.
Implementation Guidelines:
Granular Logging:
 - Implement detailed and granular logging mechanisms to capture pertinent information about API requests, responses, user authentication, and authorization events.
 - Log relevant metadata such as timestamps, source IP addresses, user identities, and the nature of the API transactions.**Sensitive Data Masking:**
 - Apply appropriate data masking techniques to prevent the logging of sensitive information such as passwords, personal identifiable information (PII), and other confidential data.**Logging Standards:**
 - Adhere to industry-standard logging formats (e.g., JSON, syslog) to ensure compatibility with SIEM solutions and facilitate efficient log parsing and analysis.**Log Retention and Rotation:**
 - Define a log retention policy to ensure logs are retained for an appropriate duration based on compliance requirements and security best practices.
 - Implement log rotation to prevent storage overflow and maintain a manageable log volume.**Real-time Alerting:**
 - Integrate the logging system with a SIEM solution to enable real-time monitoring and alerting on suspicious or anomalous activities.
 - Configure alerts for specific events that may indicate security incidents, such as multiple failed login attempts or unusual patterns of API access.**Access Controls for Logs:**
 - Implement access controls and encryption for log files to restrict access only to authorized personnel.
 - Regularly review and audit access to log files to ensure compliance and detect unauthorized access.**Integration with Incident Response:**
 - Ensure that the logging system is seamlessly integrated with the organization's incident response processes and tools.
 - Define clear procedures for incident identification, analysis, and response based on the information gathered from the logs.**Regular Monitoring and Review:**
 - Periodically review and analyze logs to identify trends, potential security threats, or areas for improvement in the API service's security posture.
 - Conduct regular audits of the logging and SIEM integration to validate the effectiveness of the control.**Benefits:**
 - Early detection of security incidents.
 - Improved forensic capabilities for investigating security breaches.
 - Compliance with regulatory requirements regarding data protection and incident reporting.
 - Enhanced visibility into API activities for both security and operational purposes.**Note to Developers:** Implementing comprehensive logging and SIEM integration is not just a security requirement but a crucial aspect of maintaining the integrity and reliability of our API Service. Follow the guidelines diligently and collaborate with the security team to ensure the effective implementation of this control.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

- Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High
- Rec2. Block or validate all outbound requests C-WEB-APPLICATION-SERVER-SIDE-CNT-11 Medium
- Rec3. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09 Medium
- Rec4. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12 Very high

Component: API Service

Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high

PCI-DSS-v3.2.1: 11.1


Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

3 Recommended countermeasures

Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High ☐ Not tested

- State:  Recommended
 - Description:
To minimize the attack surface and enhance security, disable any unused services, remove default accounts, and apply security updates promptly. By doing so, you reduce the risk of exploitation through unneeded services or default credentials and ensure that the system is protected from known vulnerabilities. Additionally, use secure configurations to prevent unauthorized access and ensure that the system is optimally protected.
Implementation Steps:
 1. **Disable Unused Services:** Identify and disable any services that are not needed for the application or system to function. This limits the number of potential attack vectors.
 2. **Remove Default Accounts:** Remove or disable default accounts and ensure that all active accounts are assigned strong, unique credentials. Avoid using default settings that might be easily guessed by attackers.
 3. **Apply Security Updates:** Regularly check for and apply security patches to all systems and applications. Set up an automated patch management process to ensure that critical updates are applied as soon as they are released.
 4. **Use Secure Configurations:** Ensure that all configurations, such as database, server, and application settings, follow security best practices, including strong encryption, least privilege access, and secure communication protocols.
- References:

- [OWASP Secure Configuration Guide](#)

Rec2. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09 Medium Not tested

- State: Recommended
- Description:

To ensure comprehensive visibility into system activities, capture detailed logs for all critical actions within the web application. These logs should include data such as user actions, system changes, and access to sensitive resources. Additionally, proactively monitor these logs to detect suspicious behavior, unauthorized access, or anomalous activity, enabling timely detection of security incidents and faster response to potential threats.

Implementation Steps:

 1. **Enable Logging for Critical Actions:** Ensure that all key actions (e.g., login attempts, data access, configuration changes) are logged in detail, including relevant metadata such as timestamps, user identifiers, and source IP addresses.
 2. **Use Centralized Logging:** Store logs in a centralized location for easier access and analysis. Integrate with a log management or SIEM system for real-time monitoring.
 3. **Set Up Automated Alerts:** Configure automated alerts for suspicious or unauthorized actions, such as failed login attempts, unexpected configuration changes, or access to sensitive data by unauthorized users.
 4. **Regularly Review Logs:** Conduct regular log reviews and audits to ensure that logging mechanisms are functioning correctly and that logs provide the necessary level of detail to identify security incidents.

References:

 - [OWASP Logging Best Practices](#)

Rec3. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12 Very high Not tested

- State: Recommended
- Description:

To prevent workflow abuses, especially for critical or sensitive transactions, implement server-side checks and multi-step validation. This ensures that transactions are properly validated at multiple points in the process, reducing the risk of unauthorized actions, fraud, or exploitation of business workflows.

Implementation Steps:

 1. **Define Critical Transactions:** Identify which transactions are critical (e.g., financial transactions, user permissions changes) and require extra validation to ensure security.
 2. **Server-Side Validation:** Ensure that all business rules, permissions, and workflows are validated on the server side, rather than relying solely on client-side validation.
 3. **Multi-Step Validation Process:** Implement a multi-step validation process for important transactions, requiring approval or confirmation at multiple stages (e.g., email confirmation, admin approval, CAPTCHA verification).
 4. **Audit Trail:** Maintain a detailed audit trail for critical transactions, logging every step and any changes made, so that any suspicious activity can be detected and reviewed.
 5. **Review and Update Validation Rules Regularly:** Periodically review and update the validation rules to adapt to new business requirements and emerging threats.

References:

 - [PCI DSS Quick Reference Guide](#)

Component: API Service

1 Recommended countermeasures

Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high Not tested

- State: Recommended
- Description:

In order to fortify the security posture of our API Service, it is imperative to implement a robust logging mechanism and seamlessly integrate it with a Security Information and Event Management (SIEM) system. This control ensures that all relevant activities and events within the API ecosystem are recorded, monitored, and analyzed in real-time, enhancing our ability to detect and respond to security incidents promptly.

Implementation Guidelines:

Granular Logging:

 - Implement detailed and granular logging mechanisms to capture pertinent information about API requests, responses, user authentication, and authorization events.
 - Log relevant metadata such as timestamps, source IP addresses, user identities, and the nature of the API transactions.

Sensitive Data Masking:

 - Apply appropriate data masking techniques to prevent the logging of sensitive information such as passwords, personal identifiable information (PII), and other confidential data.

Logging Standards:

 - Adhere to industry-standard logging formats (e.g., JSON, syslog) to ensure compatibility with SIEM solutions and facilitate efficient log parsing and analysis.

Log Retention and Rotation:

 - Define a log retention policy to ensure logs are retained for an appropriate duration based on compliance requirements and security best practices.
 - Implement log rotation to prevent storage overflow and maintain a manageable log volume.

Real-time Alerting:

 - Integrate the logging system with a SIEM solution to enable real-time monitoring and alerting on suspicious or anomalous activities.
 - Configure alerts for specific events that may indicate security incidents, such as multiple failed login attempts or unusual patterns of API access.

Access Controls for Logs:

 - Implement access controls and encryption for log files to restrict access only to authorized personnel.
 - Regularly review and audit access to log files to ensure compliance and detect unauthorized access.

Integration with Incident Response:

 - Ensure that the logging system is seamlessly integrated with the organization's incident response processes and tools.
 - Define clear procedures for incident identification, analysis, and response based on the information gathered from the logs.

Regular Monitoring and Review:

 - Periodically review and analyze logs to identify trends, potential security threats, or areas for improvement in the API service's security posture.
 - Conduct regular audits of the logging and SIEM integration to validate the effectiveness of the control.

Benefits:

 - Early detection of security incidents.
 - Improved forensic capabilities for investigating security breaches.
 - Compliance with regulatory requirements regarding data protection and incident reporting.
 - Enhanced visibility into API activities for both security and operational purposes.

Note to Developers: Implementing comprehensive logging and SIEM integration is not just a security requirement but a crucial aspect of maintaining the integrity and reliability of our API Service. Follow the guidelines diligently and collaborate with the security team to ensure the effective implementation of this control.

Component: Document-Oriented Database

1 Recommended countermeasures

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CMS Very high Not tested

- State: Recommended
- Description:

Conducting regular configuration audits of a Document-Oriented Database is vital for ensuring that security settings and configurations are aligned with best practices and organizational security policies. Here's how to systematically approach these audits:

 1. Establish a Baseline Configuration
 - **Identify Security Settings:** Document the optimal security configurations for your database environment. This includes access controls, encryption settings, network configurations, and any specific database security features.
 - **Create a Configuration Template:** Develop a template or checklist that outlines all security-related configuration settings. This template will serve as the baseline for regular audits.
 2. Automate the Audit Process

- **Use Configuration Management Tools:** Implement tools that can automate the process of detecting deviations from the baseline configuration. Tools like Puppet, Chef, Ansible, or database-specific management solutions can help.
 - **Schedule Regular Scans:** Configure these tools to perform regular scans of your document-oriented database environment to identify and report any configuration changes or non-compliance with the baseline.
3. Review Access Controls
- **Validate User Access:** Regularly review user accounts, roles, and permissions to ensure that they adhere to the principle of least privilege. Remove or adjust any excessive permissions.
 - **Audit Authentication Mechanisms:** Verify that authentication methods (password policies, multi-factor authentication, etc.) are properly configured and enforced.
4. Verify Network Configurations
- **Check Network Access Controls:** Ensure that the database is not exposed to unauthorized networks. Validate firewall rules and other network security configurations that control access to the database.
 - **Secure Communication Channels:** Confirm that data in transit is encrypted and that secure communication protocols are in use.
5. Confirm Data Protection Measures
- **Data Encryption:** Verify that data at rest is encrypted according to best practices. Check the configuration of encryption mechanisms and the management of encryption keys.
 - **Backup and Recovery:** Review the configuration of backup processes to ensure that data is securely backed up and that encryption is maintained for backups.
6. Document and Resolve Findings
- **Report Findings:** Document any deviations from the baseline configuration identified during the audit. Classify findings based on their potential impact on security.
 - **Plan Remediation:** Develop a plan for addressing high-priority findings. This may include reconfiguring settings, updating software, or modifying access controls.
 - **Track Progress:** Maintain records of audits, findings, and remediation efforts to track improvements over time and to provide accountability.
7. Update Baseline Configurations
- **Incorporate Changes:** As you make changes to your database environment or as new security best practices emerge, update your baseline configuration and templates accordingly.
 - **Review Periodically:** Regularly review the baseline configuration to ensure it remains relevant and effective in mitigating security risks.
8. Educate and Train
- **Team Awareness:** Ensure that your team is aware of the configuration audit process, the importance of adhering to the baseline configurations, and the role they play in maintaining database security.
 - **Continuous Learning:** Stay informed about new security features offered by your document-oriented database system and incorporate relevant features into your security baseline.
- References and Good Security Practices**
- **Industry Best Practices:** Follow guidelines and security checklists provided by the database vendor, as well as security standards relevant to your industry (e.g., CIS Benchmarks, NIST Guidelines).
 - **Security Communities:** Engage with security communities and forums related to your database technology to stay updated on the latest security advisories and recommendations.
- Regular configuration audits are a proactive step in maintaining the security and integrity of your document-oriented database, helping to prevent unauthorized access, data leaks, and ensuring compliance with data protection regulations.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

- Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High
- Rec2. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09 Medium
- Rec3. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12 Very high

Component: API Service

- Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high

Component: Document-Oriented Database

- Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CMS Very high

PCI-DSS-v3.2.1: 11.4

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

4 Recommended countermeasures

- Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High Not tested
- State: Recommended
- Description:
- To minimize the attack surface and enhance security, disable any unused services, remove default accounts, and apply security updates promptly. By doing so, you reduce the risk of exploitation through unneeded services or default credentials and ensure that the system is protected from known vulnerabilities. Additionally, use secure configurations to prevent unauthorized access and ensure that the system is optimally protected.
- Implementation Steps:**
1. **Disable Unused Services:** Identify and disable any services that are not needed for the application or system to function. This limits the number of potential attack vectors.
 2. **Remove Default Accounts:** Remove or disable default accounts and ensure that all active accounts are assigned strong, unique credentials. Avoid using default settings that might be easily guessed by attackers.
 3. **Apply Security Updates:** Regularly check for and apply security patches to all systems and applications. Set up an automated patch management process to ensure that critical updates are applied as soon as they are released.
 4. **Use Secure Configurations:** Ensure that all configurations, such as database, server, and application settings, follow security best practices, including strong encryption, least privilege access, and secure communication protocols.
- References:**
- [OWASP Secure Configuration Guide](#)
- Rec2. Block or validate all outbound requests C-WEB-APPLICATION-SERVER-SIDE-CNT-11 Medium Not tested
- State: Recommended
- Description:
- To reduce the risk of data exfiltration or unauthorized system communication, block or carefully validate all outbound requests made by the application. Ensure that only requests to specific, trusted domains and IP addresses are allowed, and that any outbound request to unapproved destinations is blocked or flagged for further review. This control prevents the application from communicating with malicious external systems and protects sensitive data from being leaked.
- Implementation Steps:**
1. **Implement Outbound Request Filtering:** Use a web application firewall (WAF) or proxy to block or monitor any outbound requests that are not to trusted domains or IP addresses.
 2. **Validate Outbound Requests:** Before allowing outbound communication, validate the destination domain and IP address to ensure they are within an approved list of resources necessary for the application.

- 3. **Set Domain and IP Allowlists:** Define an allowlist of specific domains and IP addresses that the application is authorized to interact with, and block all other outbound traffic.
- 4. **Monitor and Log Outbound Requests:** Continuously monitor and log all outbound requests for suspicious activity or attempts to communicate with unapproved destinations.
- 5. **Review and Update Allowlist Regularly:** Periodically review the allowlist to ensure that it remains up-to-date with the application's legitimate requirements.

References:

- [Security and Privacy Controls for Information Systems and Organizations](#)

Rec3. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09 Medium ☐ Not tested

- State: Recommended
- Description:

To ensure comprehensive visibility into system activities, capture detailed logs for all critical actions within the web application. These logs should include data such as user actions, system changes, and access to sensitive resources. Additionally, proactively monitor these logs to detect suspicious behavior, unauthorized access, or anomalous activity, enabling timely detection of security incidents and faster response to potential threats.

Implementation Steps:

1. **Enable Logging for Critical Actions:** Ensure that all key actions (e.g., login attempts, data access, configuration changes) are logged in detail, including relevant metadata such as timestamps, user identifiers, and source IP addresses.
2. **Use Centralized Logging:** Store logs in a centralized location for easier access and analysis. Integrate with a log management or SIEM system for real-time monitoring.
3. **Set Up Automated Alerts:** Configure automated alerts for suspicious or unauthorized actions, such as failed login attempts, unexpected configuration changes, or access to sensitive data by unauthorized users.
4. **Regularly Review Logs:** Conduct regular log reviews and audits to ensure that logging mechanisms are functioning correctly and that logs provide the necessary level of detail to identify security incidents.

References:

- [OWASP Logging Best Practices](#)

Rec4. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12 Very high ☐ Not tested

- State: Recommended
- Description:

To prevent workflow abuses, especially for critical or sensitive transactions, implement server-side checks and multi-step validation. This ensures that transactions are properly validated at multiple points in the process, reducing the risk of unauthorized actions, fraud, or exploitation of business workflows.

Implementation Steps:

1. **Define Critical Transactions:** Identify which transactions are critical (e.g., financial transactions, user permissions changes) and require extra validation to ensure security.
2. **Server-Side Validation:** Ensure that all business rules, permissions, and workflows are validated on the server side, rather than relying solely on client-side validation.
3. **Multi-Step Validation Process:** Implement a multi-step validation process for important transactions, requiring approval or confirmation at multiple stages (e.g., email confirmation, admin approval, CAPTCHA verification).
4. **Audit Trail:** Maintain a detailed audit trail for critical transactions, logging every step and any changes made, so that any suspicious activity can be detected and reviewed.
5. **Review and Update Validation Rules Regularly:** Periodically review and update the validation rules to adapt to new business requirements and emerging threats.

References:

- [PCI DSS Quick Reference Guide](#)

Component: API Service

1 Recommended countermeasures

Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high ☐ Not tested

- State: Recommended
- Description:

In order to fortify the security posture of our API Service, it is imperative to implement a robust logging mechanism and seamlessly integrate it with a Security Information and Event Management (SIEM) system. This control ensures that all relevant activities and events within the API ecosystem are recorded, monitored, and analyzed in real-time, enhancing our ability to detect and respond to security incidents promptly.

Implementation Guidelines:

Granular Logging:

- Implement detailed and granular logging mechanisms to capture pertinent information about API requests, responses, user authentication, and authorization events.
- Log relevant metadata such as timestamps, source IP addresses, user identities, and the nature of the API transactions.

Sensitive Data Masking:

- Apply appropriate data masking techniques to prevent the logging of sensitive information such as passwords, personal identifiable information (PII), and other confidential data.

Logging Standards:

- Adhere to industry-standard logging formats (e.g., JSON, syslog) to ensure compatibility with SIEM solutions and facilitate efficient log parsing and analysis.

Log Retention and Rotation:

- Define a log retention policy to ensure logs are retained for an appropriate duration based on compliance requirements and security best practices.
- Implement log rotation to prevent storage overflow and maintain a manageable log volume.

Real-time Alerting:

- Integrate the logging system with a SIEM solution to enable real-time monitoring and alerting on suspicious or anomalous activities.
- Configure alerts for specific events that may indicate security incidents, such as multiple failed login attempts or unusual patterns of API access.

Access Controls for Logs:

- Implement access controls and encryption for log files to restrict access only to authorized personnel.
- Regularly review and audit access to log files to ensure compliance and detect unauthorized access.

Integration with Incident Response:

- Ensure that the logging system is seamlessly integrated with the organization's incident response processes and tools.
- Define clear procedures for incident identification, analysis, and response based on the information gathered from the logs.

Regular Monitoring and Review:

- Periodically review and analyze logs to identify trends, potential security threats, or areas for improvement in the API service's security posture.
- Conduct regular audits of the logging and SIEM integration to validate the effectiveness of the control.

Benefits:

- Early detection of security incidents.
- Improved forensic capabilities for investigating security breaches.
- Compliance with regulatory requirements regarding data protection and incident reporting.
- Enhanced visibility into API activities for both security and operational purposes.

Note to Developers: Implementing comprehensive logging and SIEM integration is not just a security requirement but a crucial aspect of maintaining the integrity and reliability of our API Service. Follow the guidelines diligently and collaborate with the security team to ensure the effective implementation of this control.

Component: Document-Oriented Database

1 Recommended countermeasures

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CMS Very high ☐ Not tested

- State: Recommended
- Description:

Conducting regular configuration audits of a Document-Oriented Database is vital for ensuring that security settings and configurations are aligned with best practices and organizational security policies. Here's how to systematically approach these audits:

1. Establish a Baseline Configuration

- **Identify Security Settings:** Document the optimal security configurations for your database environment. This includes access controls, encryption settings, network configurations, and any specific database security features.
 - **Create a Configuration Template:** Develop a template or checklist that outlines all security-related configuration settings. This template will serve as the baseline for regular audits.
2. Automate the Audit Process
- **Use Configuration Management Tools:** Implement tools that can automate the process of detecting deviations from the baseline configuration. Tools like Puppet, Chef, Ansible, or database-specific management solutions can help.
 - **Schedule Regular Scans:** Configure these tools to perform regular scans of your document-oriented database environment to identify and report any configuration changes or non-compliance with the baseline.
3. Review Access Controls
- **Validate User Access:** Regularly review user accounts, roles, and permissions to ensure that they adhere to the principle of least privilege. Remove or adjust any excessive permissions.
 - **Audit Authentication Mechanisms:** Verify that authentication methods (password policies, multi-factor authentication, etc.) are properly configured and enforced.
4. Verify Network Configurations
- **Check Network Access Controls:** Ensure that the database is not exposed to unauthorized networks. Validate firewall rules and other network security configurations that control access to the database.
 - **Secure Communication Channels:** Confirm that data in transit is encrypted and that secure communication protocols are in use.
5. Confirm Data Protection Measures
- **Data Encryption:** Verify that data at rest is encrypted according to best practices. Check the configuration of encryption mechanisms and the management of encryption keys.
 - **Backup and Recovery:** Review the configuration of backup processes to ensure that data is securely backed up and that encryption is maintained for backups.
6. Document and Resolve Findings
- **Report Findings:** Document any deviations from the baseline configuration identified during the audit. Classify findings based on their potential impact on security.
 - **Plan Remediation:** Develop a plan for addressing high-priority findings. This may include reconfiguring settings, updating software, or modifying access controls.
 - **Track Progress:** Maintain records of audits, findings, and remediation efforts to track improvements over time and to provide accountability.
7. Update Baseline Configurations
- **Incorporate Changes:** As you make changes to your database environment or as new security best practices emerge, update your baseline configuration and templates accordingly.
 - **Review Periodically:** Regularly review the baseline configuration to ensure it remains relevant and effective in mitigating security risks.
8. Educate and Train
- **Team Awareness:** Ensure that your team is aware of the configuration audit process, the importance of adhering to the baseline configurations, and the role they play in maintaining database security.
 - **Continuous Learning:** Stay informed about new security features offered by your document-oriented database system and incorporate relevant features into your security baseline.
- References and Good Security Practices**
- **Industry Best Practices:** Follow guidelines and security checklists provided by the database vendor, as well as security standards relevant to your industry (e.g., CIS Benchmarks, NIST Guidelines).
 - **Security Communities:** Engage with security communities and forums related to your database technology to stay updated on the latest security advisories and recommendations.
- Regular configuration audits are a proactive step in maintaining the security and integrity of your document-oriented database, helping to prevent unauthorized access, data leaks, and ensuring compliance with data protection regulations.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

- Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High
- Rec2. Block or validate all outbound requests C-WEB-APPLICATION-SERVER-SIDE-CNT-11 Medium
- Rec3. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09 Medium
- Rec4. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12 Very high

Component: API Service

- Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high

Component: Document-Oriented Database

- Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CM5 Very high

PCI-DSS-v3.2.1: 11.5

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

7 Recommended countermeasures

- Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High Not tested
- **State:** Recommended
 - **Description:**

To minimize the attack surface and enhance security, disable any unused services, remove default accounts, and apply security updates promptly. By doing so, you reduce the risk of exploitation through unneeded services or default credentials and ensure that the system is protected from known vulnerabilities. Additionally, use secure configurations to prevent unauthorized access and ensure that the system is optimally protected.

Implementation Steps:

 1. **Disable Unused Services:** Identify and disable any services that are not needed for the application or system to function. This limits the number of potential attack vectors.
 2. **Remove Default Accounts:** Remove or disable default accounts and ensure that all active accounts are assigned strong, unique credentials. Avoid using default settings that might be easily guessed by attackers.
 3. **Apply Security Updates:** Regularly check for and apply security patches to all systems and applications. Set up an automated patch management process to ensure that critical updates are applied as soon as they are released.
 4. **Use Secure Configurations:** Ensure that all configurations, such as database, server, and application settings, follow security best practices, including strong encryption, least privilege access, and secure communication protocols.
 - **References:**
 - [OWASP Secure Configuration Guide](#)
- Rec2. Ensure proper escaping/encoding of dynamic content and configure CSP C-WEB-APPLICATION-SERVER-SIDE-CNT-04 High Not tested
- **State:** Recommended
 - **Description:**

To mitigate the risk of script injection attacks such as Cross-Site Scripting (XSS), ensure that all dynamic content in the web application is properly escaped or encoded before being rendered. This prevents malicious scripts from being executed in the user's browser. Additionally, configure a Content Security Policy (CSP) to limit the sources of executable content, reducing the risk of inline script execution and script-based attacks.

Implementation Steps:

- 1. **Escape or Encode Dynamic Content:** Ensure that any dynamic content included in web pages (e.g., user inputs, data from external sources) is properly escaped or encoded to prevent execution of potentially harmful scripts.
- 2. **Use CSP to Control Script Sources:** Configure a strict Content Security Policy (CSP) to restrict which domains can serve executable content (e.g., scripts, styles). For example, disallow inline scripts and only allow scripts from trusted domains.
- 3. **Monitor and Update CSP:** Regularly review and update the CSP to ensure it is correctly configured as new resources or third-party services are added to the application.
- 4. **Test for XSS Vulnerabilities:** Conduct regular security testing to ensure that content is correctly sanitized and that the CSP is enforced.

References:

- [OWASP XSS Prevention Cheat Sheet](#)

Rec3. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09 Medium ☐ Not tested

- State: Recommended
- Description:

To ensure comprehensive visibility into system activities, capture detailed logs for all critical actions within the web application. These logs should include data such as user actions, system changes, and access to sensitive resources. Additionally, proactively monitor these logs to detect suspicious behavior, unauthorized access, or anomalous activity, enabling timely detection of security incidents and faster response to potential threats.

Implementation Steps:

 1. **Enable Logging for Critical Actions:** Ensure that all key actions (e.g., login attempts, data access, configuration changes) are logged in detail, including relevant metadata such as timestamps, user identifiers, and source IP addresses.
 2. **Use Centralized Logging:** Store logs in a centralized location for easier access and analysis. Integrate with a log management or SIEM system for real-time monitoring.
 3. **Set Up Automated Alerts:** Configure automated alerts for suspicious or unauthorized actions, such as failed login attempts, unexpected configuration changes, or access to sensitive data by unauthorized users.
 4. **Regularly Review Logs:** Conduct regular log reviews and audits to ensure that logging mechanisms are functioning correctly and that logs provide the necessary level of detail to identify security incidents.

References:

- [OWASP Logging Best Practices](#)

Rec4. Implement prepared statements and validate user inputs C-WEB-APPLICATION-SERVER-SIDE-CNT-01 High ☐ Not tested

- State: Recommended
- Description:

To prevent injection attacks (e.g., SQL injection, command injection), implement prepared statements for all database queries and validate all user inputs on the server side. This ensures that user input is treated as data, not executable code, and prevents attackers from injecting malicious commands that could compromise your application or database.

Implementation Steps:

 1. **Use Prepared Statements:** Always use prepared statements with parameterized queries for interacting with the database, ensuring that user inputs are never directly included in SQL queries.
 2. **Sanitize and Validate Inputs:** Validate all incoming data on the server side (e.g., using whitelist validation or data type checking) to ensure it conforms to expected formats and ranges.
 3. **Escape Output:** Properly escape or encode output to prevent cross-site scripting (XSS) and other injection attacks when displaying user-provided data.
 4. **Use ORM or Query Builders:** If possible, use Object-Relational Mapping (ORM) frameworks or query builders that automatically handle safe query construction.
 5. **Implement Content Security Policies:** Use security controls like Content Security Policies (CSP) and Input Validation to further mitigate injection risks.

References:

- [OWASP SQL Injection Prevention Cheat Sheet](#)

Rec5. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12 Very high ☐ Not tested

- State: Recommended
- Description:

To prevent workflow abuses, especially for critical or sensitive transactions, implement server-side checks and multi-step validation. This ensures that transactions are properly validated at multiple points in the process, reducing the risk of unauthorized actions, fraud, or exploitation of business workflows.

Implementation Steps:

 1. **Define Critical Transactions:** Identify which transactions are critical (e.g., financial transactions, user permissions changes) and require extra validation to ensure security.
 2. **Server-Side Validation:** Ensure that all business rules, permissions, and workflows are validated on the server side, rather than relying solely on client-side validation.
 3. **Multi-Step Validation Process:** Implement a multi-step validation process for important transactions, requiring approval or confirmation at multiple stages (e.g., email confirmation, admin approval, CAPTCHA verification).
 4. **Audit Trail:** Maintain a detailed audit trail for critical transactions, logging every step and any changes made, so that any suspicious activity can be detected and reviewed.
 5. **Review and Update Validation Rules Regularly:** Periodically review and update the validation rules to adapt to new business requirements and emerging threats.

References:

- [PCI DSS Quick Reference Guide](#)

Rec6. Perform software composition analysis and patch outdated libraries C-WEB-APPLICATION-SERVER-SIDE-CNT-08 Very high ☐ Not tested

- State: Recommended
 - Description:

To reduce the risk of vulnerabilities from outdated or insecure libraries, perform software composition analysis (SCA) to identify all third-party libraries and dependencies used in your web application. Regularly scan for known vulnerabilities and outdated components, then promptly patch or upgrade them to secure versions. This ensures that your application is protected from known exploits related to insecure or deprecated libraries.

Implementation Steps:

 1. **Perform Software Composition Analysis:** Use automated tools to scan your codebase and identify all third-party libraries and dependencies, including their versions.
 2. **Identify Vulnerabilities:** Cross-reference your libraries against known vulnerability databases (e.g., CVE, NVD) to identify any libraries that are outdated or have known security issues.
 3. **Patch or Upgrade Libraries:** Once outdated or vulnerable libraries are identified, promptly upgrade to the latest stable versions or apply necessary patches to mitigate any security risks.
 4. **Automate Regular Scanning:** Set up automated scanning processes to regularly check for vulnerabilities in your dependencies, ensuring that any new risks are identified and addressed in a timely manner.
- References:
- [OWASP Software Composition Analysis](#)

Rec7. Use secure data formats and strict whitelisting for deserialization C-WEB-APPLICATION-SERVER-SIDE-CNT-07 Medium ☐ Not tested

- State: Recommended
 - Description:

To prevent deserialization vulnerabilities, ensure that your web application uses secure data formats, such as JSON, and applies strict whitelisting to limit the types of objects that can be deserialized. This minimizes the risk of malicious code execution or object manipulation during deserialization. By enforcing tight control over deserialized data and limiting the allowed object types, you reduce the attack surface and prevent attackers from injecting malicious objects that could compromise the server.

Implementation Steps:

 1. **Use Secure Data Formats:** Always prefer secure, text-based formats like JSON over binary formats to reduce the risk of deserialization attacks.
 2. **Implement Whitelisting:** Configure deserialization routines to only allow specific, trusted object types. Any unsupported object type should be rejected immediately.
 3. **Sanitize Input:** Apply rigorous validation and sanitization to all input before deserialization to ensure it does not contain harmful data.
 4. **Monitor for Malicious Input:** Continuously monitor incoming data for patterns indicating potential deserialization attacks, such as malformed objects.
- References:
- [OWASP Deserialization Cheat Sheet](#)

Component: CLI Application

2 Recommended countermeasures

Rec1. Implement comprehensive input validation and sanitization C-CLI-APPLICATION-C005 High ☐ Not tested

- State: Recommended
- Description:

To safeguard your CLI application against security threats such as unauthorized command execution or code injection, implementing robust input validation and sanitization is paramount. This process involves several key strategies designed to scrutinize and clean all user-supplied data before it's processed by your application. By adhering to the following practices, developers can significantly reduce the application's vulnerability to malicious input:

 - Whitelisting:** Define a strict list of acceptable inputs for each field or parameter your application uses. Only allow inputs that match the predefined criteria, effectively blocking unexpected or potentially harmful data. Whitelisting is preferable to blacklisting, as it ensures that only known safe inputs are accepted.
 - Input Filtering:** Employ input filtering mechanisms to analyze and clean data before processing. This can involve stripping out or replacing characters and patterns that are commonly used in injection attacks. Regular expressions can be useful for this purpose, but they must be used cautiously to avoid inadvertently allowing dangerous inputs.
 - Secure Coding Practices:** Follow secure coding guidelines that emphasize input validation and sanitization. This includes using prepared statements or parameterized queries when interacting with databases to prevent SQL injection, and encoding data before output to prevent cross-site scripting (XSS) attacks.
 - Length and Type Checks:** Perform length and type checks on all user-supplied data. Ensure that the input length does not exceed expected ranges and that the data type matches what is required for processing. This can prevent buffer overflow attacks and other exploits targeting data type mismatches.
 - Error Handling:** Implement secure error handling that does not disclose sensitive information to the user. In cases where input validation fails, provide generic error messages that do not reveal details about the application's internal workings or database structure.

Rec2. Implement URL validation and sanitization C-CLI-APPLICATION-C004 Medium ☐ Not tested

- State: Recommended
- Description:

In the development and maintenance of Command-Line Interface (CLI) Applications, particularly those interacting with web resources, it's crucial to mitigate risks associated with unvalidated or unsanitized URL inputs. Maliciously crafted URLs can lead to a variety of security vulnerabilities, including but not limited to, redirection attacks, command injection, and exposure to malicious sites. To protect your application and its users, implementing rigorous URL validation and sanitization processes is essential. This involves:

 - Strict Validation:** Before processing any URLs within your CLI application, validate them rigorously to ensure they conform to a predefined format or pattern. Employ regular expressions or URL parsing libraries to verify the structure of the URLs and ensure they are syntactically correct.
 - Sanitization:** Sanitize all URL inputs by encoding or removing potentially dangerous characters that could be exploited for injection attacks. Ensure that parameters within the URLs are also properly encoded to prevent cross-site scripting (XSS) and other injection vulnerabilities.
 - Trusted Destinations:** Maintain a whitelist (allowlist) of trusted destinations. Only allow redirects and forwards to URLs within this predefined list to prevent unvalidated redirect and forward attacks, where users could be directed to phishing sites or other malicious web resources.
 - Use of Secure Libraries:** Utilize well-maintained and secure libraries for URL validation and sanitization. These libraries often provide more comprehensive and up-to-date protection against common and emerging threats associated with URL processing.
 - Regular Updates:** Keep your URL validation and sanitization mechanisms, as well as any associated libraries, regularly updated. This ensures that your CLI application is protected against newly discovered vulnerabilities and attack techniques.

Component: API Service

1 Recommended countermeasures

Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high ☐ Not tested

- State: Recommended
- Description:

In order to fortify the security posture of our API Service, it is imperative to implement a robust logging mechanism and seamlessly integrate it with a Security Information and Event Management (SIEM) system. This control ensures that all relevant activities and events within the API ecosystem are recorded, monitored, and analyzed in real-time, enhancing our ability to detect and respond to security incidents promptly.

Implementation Guidelines:

Granular Logging:

 - Implement detailed and granular logging mechanisms to capture pertinent information about API requests, responses, user authentication, and authorization events.
 - Log relevant metadata such as timestamps, source IP addresses, user identities, and the nature of the API transactions.

Sensitive Data Masking:

 - Apply appropriate data masking techniques to prevent the logging of sensitive information such as passwords, personal identifiable information (PII), and other confidential data.

Logging Standards:

 - Adhere to industry-standard logging formats (e.g., JSON, syslog) to ensure compatibility with SIEM solutions and facilitate efficient log parsing and analysis.

Log Retention and Rotation:

 - Define a log retention policy to ensure logs are retained for an appropriate duration based on compliance requirements and security best practices.
 - Implement log rotation to prevent storage overflow and maintain a manageable log volume.

Real-time Alerting:

 - Integrate the logging system with a SIEM solution to enable real-time monitoring and alerting on suspicious or anomalous activities.
 - Configure alerts for specific events that may indicate security incidents, such as multiple failed login attempts or unusual patterns of API access.

Access Controls for Logs:

 - Implement access controls and encryption for log files to restrict access only to authorized personnel.
 - Regularly review and audit access to log files to ensure compliance and detect unauthorized access.

Integration with Incident Response:

 - Ensure that the logging system is seamlessly integrated with the organization's incident response processes and tools.
 - Define clear procedures for incident identification, analysis, and response based on the information gathered from the logs.

Regular Monitoring and Review:

 - Periodically review and analyze logs to identify trends, potential security threats, or areas for improvement in the API service's security posture.
 - Conduct regular audits of the logging and SIEM integration to validate the effectiveness of the control.

Benefits:

 - Early detection of security incidents.
 - Improved forensic capabilities for investigating security breaches.
 - Compliance with regulatory requirements regarding data protection and incident reporting.
 - Enhanced visibility into API activities for both security and operational purposes.

Note to Developers: Implementing comprehensive logging and SIEM integration is not just a security requirement but a crucial aspect of maintaining the integrity and reliability of our API Service. Follow the guidelines diligently and collaborate with the security team to ensure the effective implementation of this control.

Component: Document-Oriented Database

1 Recommended countermeasures

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CM5 Very high ☐ Not tested

- State: Recommended
- Description:

Conducting regular configuration audits of a Document-Oriented Database is vital for ensuring that security settings and configurations are aligned with best practices and organizational security policies. Here's how to systematically approach these audits:

 - Establish a Baseline Configuration
 - Identify Security Settings:** Document the optimal security configurations for your database environment. This includes access controls, encryption settings, network configurations, and any specific database security features.

- **Create a Configuration Template:** Develop a template or checklist that outlines all security-related configuration settings. This template will serve as the baseline for regular audits.
2. Automate the Audit Process
- **Use Configuration Management Tools:** Implement tools that can automate the process of detecting deviations from the baseline configuration. Tools like Puppet, Chef, Ansible, or database-specific management solutions can help.
 - **Schedule Regular Scans:** Configure these tools to perform regular scans of your document-oriented database environment to identify and report any configuration changes or non-compliance with the baseline.
3. Review Access Controls
- **Validate User Access:** Regularly review user accounts, roles, and permissions to ensure that they adhere to the principle of least privilege. Remove or adjust any excessive permissions.
 - **Audit Authentication Mechanisms:** Verify that authentication methods (password policies, multi-factor authentication, etc.) are properly configured and enforced.
4. Verify Network Configurations
- **Check Network Access Controls:** Ensure that the database is not exposed to unauthorized networks. Validate firewall rules and other network security configurations that control access to the database.
 - **Secure Communication Channels:** Confirm that data in transit is encrypted and that secure communication protocols are in use.
5. Confirm Data Protection Measures
- **Data Encryption:** Verify that data at rest is encrypted according to best practices. Check the configuration of encryption mechanisms and the management of encryption keys.
 - **Backup and Recovery:** Review the configuration of backup processes to ensure that data is securely backed up and that encryption is maintained for backups.
6. Document and Resolve Findings
- **Report Findings:** Document any deviations from the baseline configuration identified during the audit. Classify findings based on their potential impact on security.
 - **Plan Remediation:** Develop a plan for addressing high-priority findings. This may include reconfiguring settings, updating software, or modifying access controls.
 - **Track Progress:** Maintain records of audits, findings, and remediation efforts to track improvements over time and to provide accountability.
7. Update Baseline Configurations
- **Incorporate Changes:** As you make changes to your database environment or as new security best practices emerge, update your baseline configuration and templates accordingly.
 - **Review Periodically:** Regularly review the baseline configuration to ensure it remains relevant and effective in mitigating security risks.
8. Educate and Train
- **Team Awareness:** Ensure that your team is aware of the configuration audit process, the importance of adhering to the baseline configurations, and the role they play in maintaining database security.
 - **Continuous Learning:** Stay informed about new security features offered by your document-oriented database system and incorporate relevant features into your security baseline.
- References and Good Security Practices**
- **Industry Best Practices:** Follow guidelines and security checklists provided by the database vendor, as well as security standards relevant to your industry (e.g., CIS Benchmarks, NIST Guidelines).
 - **Security Communities:** Engage with security communities and forums related to your database technology to stay updated on the latest security advisories and recommendations.
- Regular configuration audits are a proactive step in maintaining the security and integrity of your document-oriented database, helping to prevent unauthorized access, data leaks, and ensuring compliance with data protection regulations.

Component: Source Code Repository

1 Recommended countermeasures

Rec1. Layer a process of cryptographic signing for source code repositories C-SOURCE-CODE-REPOSITORY-CNT-01

Very high

Not tested

State:

Recommended

Description:

Code signing uses cryptographic hashing that can validate the authenticity of software and ensure the integrity of code by verifying it has not been tampered with since being published. Code signing plays an important role in verifying the integrity of software used or distributed by organizations and individuals.

Remediation:

Layer a process of cryptographic signing and verifying code on top of the repository which can help to increase confidence that the code has not been tampered with.

Reference:

Protect access credentials

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

- Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06

High
- Rec2. Ensure proper escaping/encoding of dynamic content and configure CSP C-WEB-APPLICATION-SERVER-SIDE-CNT-04

High
- Rec3. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09

Medium
- Rec4. Implement prepared statements and validate user inputs C-WEB-APPLICATION-SERVER-SIDE-CNT-01

High
- Rec5. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12

Very high
- Rec6. Perform software composition analysis and patch outdated libraries C-WEB-APPLICATION-SERVER-SIDE-CNT-08

Very high
- Rec7. Use secure data formats and strict whitelisting for deserialization C-WEB-APPLICATION-SERVER-SIDE-CNT-07

Medium

Component: CLI Application

- Rec1. Implement comprehensive input validation and sanitization C-CLI-APPLICATION-C005

High
- Rec2. Implement URL validation and sanitization C-CLI-APPLICATION-C004

Medium

Component: API Service

- Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM

Very high

Component: Document-Oriented Database

- Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CM5

Very high

Component: Source Code Repository

- Rec1. Layer a process of cryptographic signing for source code repositories C-SOURCE-CODE-REPOSITORY-CNT-01

Very high

PCI-DSS-v3.2.1: 12.10

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

2 Recommended countermeasures

Rec1. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09 Medium Not tested

- State: Recommended
- Description:

To ensure comprehensive visibility into system activities, capture detailed logs for all critical actions within the web application. These logs should include data such as user actions, system changes, and access to sensitive resources. Additionally, proactively monitor these logs to detect suspicious behavior, unauthorized access, or anomalous activity, enabling timely detection of security incidents and faster response to potential threats.
- Implementation Steps:
 - Enable Logging for Critical Actions:** Ensure that all key actions (e.g., login attempts, data access, configuration changes) are logged in detail, including relevant metadata such as timestamps, user identifiers, and source IP addresses.
 - Use Centralized Logging:** Store logs in a centralized location for easier access and analysis. Integrate with a log management or SIEM system for real-time monitoring.
 - Set Up Automated Alerts:** Configure automated alerts for suspicious or unauthorized actions, such as failed login attempts, unexpected configuration changes, or access to sensitive data by unauthorized users.
 - Regularly Review Logs:** Conduct regular log reviews and audits to ensure that logging mechanisms are functioning correctly and that logs provide the necessary level of detail to identify security incidents.
- References:
 - [OWASP Logging Best Practices](#)

Rec2. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12 Very high Not tested

- State: Recommended
- Description:

To prevent workflow abuses, especially for critical or sensitive transactions, implement server-side checks and multi-step validation. This ensures that transactions are properly validated at multiple points in the process, reducing the risk of unauthorized actions, fraud, or exploitation of business workflows.
- Implementation Steps:
 - Define Critical Transactions:** Identify which transactions are critical (e.g., financial transactions, user permissions changes) and require extra validation to ensure security.
 - Server-Side Validation:** Ensure that all business rules, permissions, and workflows are validated on the server side, rather than relying solely on client-side validation.
 - Multi-Step Validation Process:** Implement a multi-step validation process for important transactions, requiring approval or confirmation at multiple stages (e.g., email confirmation, admin approval, CAPTCHA verification).
 - Audit Trail:** Maintain a detailed audit trail for critical transactions, logging every step and any changes made, so that any suspicious activity can be detected and reviewed.
 - Review and Update Validation Rules Regularly:** Periodically review and update the validation rules to adapt to new business requirements and emerging threats.
- References:
 - [PCI DSS Quick Reference Guide](#)

Component: API Service

1 Recommended countermeasures

Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high Not tested

- State: Recommended
- Description:

In order to fortify the security posture of our API Service, it is imperative to implement a robust logging mechanism and seamlessly integrate it with a Security Information and Event Management (SIEM) system. This control ensures that all relevant activities and events within the API ecosystem are recorded, monitored, and analyzed in real-time, enhancing our ability to detect and respond to security incidents promptly.
- Implementation Guidelines:
 - Granular Logging:**
 - Implement detailed and granular logging mechanisms to capture pertinent information about API requests, responses, user authentication, and authorization events.
 - Log relevant metadata such as timestamps, source IP addresses, user identities, and the nature of the API transactions.
 - Sensitive Data Masking:**
 - Apply appropriate data masking techniques to prevent the logging of sensitive information such as passwords, personal identifiable information (PII), and other confidential data.
 - Logging Standards:**
 - Adhere to industry-standard logging formats (e.g., JSON, syslog) to ensure compatibility with SIEM solutions and facilitate efficient log parsing and analysis.
 - Log Retention and Rotation:**
 - Define a log retention policy to ensure logs are retained for an appropriate duration based on compliance requirements and security best practices.
 - Implement log rotation to prevent storage overflow and maintain a manageable log volume.
 - Real-time Alerting:**
 - Integrate the logging system with a SIEM solution to enable real-time monitoring and alerting on suspicious or anomalous activities.
 - Configure alerts for specific events that may indicate security incidents, such as multiple failed login attempts or unusual patterns of API access.
 - Access Controls for Logs:**
 - Implement access controls and encryption for log files to restrict access only to authorized personnel.
 - Regularly review and audit access to log files to ensure compliance and detect unauthorized access.
 - Integration with Incident Response:**
 - Ensure that the logging system is seamlessly integrated with the organization's incident response processes and tools.
 - Define clear procedures for incident identification, analysis, and response based on the information gathered from the logs.
 - Regular Monitoring and Review:**
 - Periodically review and analyze logs to identify trends, potential security threats, or areas for improvement in the API service's security posture.
 - Conduct regular audits of the logging and SIEM integration to validate the effectiveness of the control.
 - Benefits:**
 - Early detection of security incidents.
 - Improved forensic capabilities for investigating security breaches.
 - Compliance with regulatory requirements regarding data protection and incident reporting.
 - Enhanced visibility into API activities for both security and operational purposes.
- Note to Developers: Implementing comprehensive logging and SIEM integration is not just a security requirement but a crucial aspect of maintaining the integrity and reliability of our API Service. Follow the guidelines diligently and collaborate with the security team to ensure the effective implementation of this control.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09 Medium

Rec2. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12 Very high

Component: API Service

Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high

PCI-DSS-v3.2.1: 12.8

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

2 Recommended countermeasures

Rec1. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09 Medium Not tested

- State: Recommended
- Description:

To ensure comprehensive visibility into system activities, capture detailed logs for all critical actions within the web application. These logs should include data such as user actions, system changes, and access to sensitive resources. Additionally, proactively monitor these logs to detect suspicious behavior, unauthorized access, or anomalous activity, enabling timely detection of security incidents and faster response to potential threats.

Implementation Steps:

 - Enable Logging for Critical Actions:** Ensure that all key actions (e.g., login attempts, data access, configuration changes) are logged in detail, including relevant metadata such as timestamps, user identifiers, and source IP addresses.
 - Use Centralized Logging:** Store logs in a centralized location for easier access and analysis. Integrate with a log management or SIEM system for real-time monitoring.
 - Set Up Automated Alerts:** Configure automated alerts for suspicious or unauthorized actions, such as failed login attempts, unexpected configuration changes, or access to sensitive data by unauthorized users.
 - Regularly Review Logs:** Conduct regular log reviews and audits to ensure that logging mechanisms are functioning correctly and that logs provide the necessary level of detail to identify security incidents.

References:

 - [OWASP Logging Best Practices](#)

Rec2. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12 Very high Not tested

- State: Recommended
- Description:

To prevent workflow abuses, especially for critical or sensitive transactions, implement server-side checks and multi-step validation. This ensures that transactions are properly validated at multiple points in the process, reducing the risk of unauthorized actions, fraud, or exploitation of business workflows.

Implementation Steps:

 - Define Critical Transactions:** Identify which transactions are critical (e.g., financial transactions, user permissions changes) and require extra validation to ensure security.
 - Server-Side Validation:** Ensure that all business rules, permissions, and workflows are validated on the server side, rather than relying solely on client-side validation.
 - Multi-Step Validation Process:** Implement a multi-step validation process for important transactions, requiring approval or confirmation at multiple stages (e.g., email confirmation, admin approval, CAPTCHA verification).
 - Audit Trail:** Maintain a detailed audit trail for critical transactions, logging every step and any changes made, so that any suspicious activity can be detected and reviewed.
 - Review and Update Validation Rules Regularly:** Periodically review and update the validation rules to adapt to new business requirements and emerging threats.

References:

 - [PCI DSS Quick Reference Guide](#)

Component: API Service

1 Recommended countermeasures

Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM Very high Not tested

- State: Recommended
- Description:

In order to fortify the security posture of our API Service, it is imperative to implement a robust logging mechanism and seamlessly integrate it with a Security Information and Event Management (SIEM) system. This control ensures that all relevant activities and events within the API ecosystem are recorded, monitored, and analyzed in real-time, enhancing our ability to detect and respond to security incidents promptly.

Implementation Guidelines:

Granular Logging:

 - Implement detailed and granular logging mechanisms to capture pertinent information about API requests, responses, user authentication, and authorization events.
 - Log relevant metadata such as timestamps, source IP addresses, user identities, and the nature of the API transactions.

Sensitive Data Masking:

 - Apply appropriate data masking techniques to prevent the logging of sensitive information such as passwords, personal identifiable information (PII), and other confidential data.

Logging Standards:

 - Adhere to industry-standard logging formats (e.g., JSON, syslog) to ensure compatibility with SIEM solutions and facilitate efficient log parsing and analysis.

Log Retention and Rotation:

 - Define a log retention policy to ensure logs are retained for an appropriate duration based on compliance requirements and security best practices.
 - Implement log rotation to prevent storage overflow and maintain a manageable log volume.

Real-time Alerting:

 - Integrate the logging system with a SIEM solution to enable real-time monitoring and alerting on suspicious or anomalous activities.
 - Configure alerts for specific events that may indicate security incidents, such as multiple failed login attempts or unusual patterns of API access.

Access Controls for Logs:

 - Implement access controls and encryption for log files to restrict access only to authorized personnel.
 - Regularly review and audit access to log files to ensure compliance and detect unauthorized access.

Integration with Incident Response:

 - Ensure that the logging system is seamlessly integrated with the organization's incident response processes and tools.
 - Define clear procedures for incident identification, analysis, and response based on the information gathered from the logs.

Regular Monitoring and Review:

 - Periodically review and analyze logs to identify trends, potential security threats, or areas for improvement in the API service's security posture.
 - Conduct regular audits of the logging and SIEM integration to validate the effectiveness of the control.

Benefits:

 - Early detection of security incidents.
 - Improved forensic capabilities for investigating security breaches.
 - Compliance with regulatory requirements regarding data protection and incident reporting.
 - Enhanced visibility into API activities for both security and operational purposes.

Note to Developers: Implementing comprehensive logging and SIEM integration is not just a security requirement but a crucial aspect of maintaining the integrity and reliability of our API Service. Follow the guidelines diligently and collaborate with the security team to ensure the effective implementation of this control.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

- Rec1. Capture and monitor detailed logs for critical actions C-WEB-APPLICATION-SERVER-SIDE-CNT-09

Medium
- Rec2. Implement server-side checks and multi-step validation for important transactions C-WEB-APPLICATION-SERVER-SIDE-CNT-12

Very high

Component: API Service

- Rec1. Implement Comprehensive Logging and SIEM Integration C-API-SERVICE-SIEM

Very high

PCI-DSS-v3.2.1: 12.3.10

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

2 Recommended countermeasures

- Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03

High

Not tested

State:

Recommended

Description:

To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.

Implementation Steps:

1. Role-Based Access Control (RBAC):

Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.

2. Permission Validation:

For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.

3. Enforce Fine-Grained Access Control:

Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.

4. Session Management:

Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.

5. Audit Logs:

Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.

References:

OWASP Access Control Cheat Sheet

Rec2. Use TLS for communications and protect stored data with encryption C-WEB-APPLICATION-SERVER-SIDE-CNT-05

Very high

Not tested

State:

Recommended

Description:

To secure data during transmission and at rest, ensure that all communications are protected using Transport Layer Security (TLS), and that stored data is encrypted using strong encryption algorithms. Additionally, implement robust key management practices to safeguard encryption keys and ensure that only authorized users and systems can access sensitive data.

Implementation Steps:

1. Enforce TLS for All Communications:

Configure all communication channels, including APIs, web traffic, and data transfers, to use TLS (preferably TLS 1.2 or higher) to protect data in transit.

2. Encrypt Stored Data:

Use strong encryption algorithms (e.g., AES-256) to encrypt sensitive data at rest, ensuring that unauthorized users cannot access or manipulate it.

3. Implement Key Management Practices:

Use a centralized key management system (KMS) to securely generate, store, and rotate encryption keys. Enforce access controls to ensure that only authorized systems can access encryption keys.

4. Regularly Update and Rotate Keys:

Set up automated key rotation policies to periodically update encryption keys and minimize the risk of key compromise.

References:

OWASP Transport Layer Security (TLS) Cheat Sheet

Component: CLI Application

1 Recommended countermeasures

Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003

High

Not tested

State:

Recommended

Description:

To enhance the security of your CLI application, it's essential to focus on two pivotal areas: configuration management and credential management. A strategic approach to these components not only fortifies your application against unauthorized access but also ensures the safeguarding of sensitive information. Below, we detail actionable steps developers can take to implement these practices effectively:

Robust Configuration Management Process:

Begin by establishing a standardized process for configuring systems and applications. This includes predefined security settings that align with industry best practices and compliance requirements.

Regularly review and update configurations to address newly identified vulnerabilities and security threats. Utilize automated tools where possible to ensure configurations remain consistent across environments.

Document all configuration changes and maintain version control to enable audit trails and rollback capabilities in case of issues.

Regular Security Reviews:

Conduct periodic security reviews of system and application configurations to identify potential security gaps or misconfigurations.

Engage in penetration testing and vulnerability assessments to evaluate the effectiveness of current security configurations and identify areas for improvement.

Secure Credential Management:

Store credentials (e.g., passwords, API keys, secret tokens) using encrypted storage solutions. Ensure encryption keys are managed securely and access to them is tightly controlled.

Avoid hard-coding credentials in source code or configuration files. Instead, use secure vaults or environment variables for managing sensitive information.

Implement multi-factor authentication and strong password policies to enhance the security of user accounts and protect against unauthorized access.

Regularly rotate credentials and keys, especially after personnel changes or when a breach is suspected.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03

High

Rec2. Use TLS for communications and protect stored data with encryption C-WEB-APPLICATION-SERVER-SIDE-CNT-05

Very high

Report: CodeSentinel Thread Modeling

Compliance report - 2025-03-14T18:54:57.395420405Z

48 of 97

Component: CLI Application

Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High

PCI-DSS-v3.2.1: 12.3.8

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side		
2 Recommended countermeasures		
Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High Not tested		
<div><div>State: Recommended</div><div>Description:<div>To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.</div></div><div>Implementation Steps:<div><div>1. Role-Based Access Control (RBAC):</div><div>Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.</div></div><div><div>2. Permission Validation:</div><div>For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.</div></div><div><div>3. Enforce Fine-Grained Access Control:</div><div>Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.</div></div><div><div>4. Session Management:</div><div>Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.</div></div><div><div>5. Audit Logs:</div><div>Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.</div></div></div><div>References:<div><div>OWASP Access Control Cheat Sheet</div></div></div></div>		
Rec2. Use TLS for communications and protect stored data with encryption C-WEB-APPLICATION-SERVER-SIDE-CNT-05 Very high Not tested		
<div><div>State: Recommended</div><div>Description:<div>To secure data during transmission and at rest, ensure that all communications are protected using Transport Layer Security (TLS), and that stored data is encrypted using strong encryption algorithms. Additionally, implement robust key management practices to safeguard encryption keys and ensure that only authorized users and systems can access sensitive data.</div></div><div>Implementation Steps:<div><div>1. Enforce TLS for All Communications:</div><div>Configure all communication channels, including APIs, web traffic, and data transfers, to use TLS (preferably TLS 1.2 or higher) to protect data in transit.</div></div><div><div>2. Encrypt Stored Data:</div><div>Use strong encryption algorithms (e.g., AES-256) to encrypt sensitive data at rest, ensuring that unauthorized users cannot access or manipulate it.</div></div><div><div>3. Implement Key Management Practices:</div><div>Use a centralized key management system (KMS) to securely generate, store, and rotate encryption keys. Enforce access controls to ensure that only authorized systems can access encryption keys.</div></div><div><div>4. Regularly Update and Rotate Keys:</div><div>Set up automated key rotation policies to periodically update encryption keys and minimize the risk of key compromise.</div></div></div><div>References:<div><div>OWASP Transport Layer Security (TLS) Cheat Sheet</div></div></div></div>		
Component: CLI Application		
1 Recommended countermeasures		
Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High Not tested		
<div><div>State: Recommended</div><div>Description:<div>To enhance the security of your CLI application, it's essential to focus on two pivotal areas: configuration management and credential management. A strategic approach to these components not only fortifies your application against unauthorized access but also ensures the safeguarding of sensitive information. Below, we detail actionable steps developers can take to implement these practices effectively:</div></div><div><div>Robust Configuration Management Process:</div><div><div>Begin by establishing a standardized process for configuring systems and applications. This includes predefined security settings that align with industry best practices and compliance requirements.</div><div>Regularly review and update configurations to address newly identified vulnerabilities and security threats. Utilize automated tools where possible to ensure configurations remain consistent across environments.</div><div>Document all configuration changes and maintain version control to enable audit trails and rollback capabilities in case of issues.</div></div><div><div>Regular Security Reviews:</div><div><div>Conduct periodic security reviews of system and application configurations to identify potential security gaps or misconfigurations.</div><div>Engage in penetration testing and vulnerability assessments to evaluate the effectiveness of current security configurations and identify areas for improvement.</div></div><div><div>Secure Credential Management:</div><div><div>Store credentials (e.g., passwords, API keys, secret tokens) using encrypted storage solutions. Ensure encryption keys are managed securely and access to them is tightly controlled.</div><div>Avoid hard-coding credentials in source code or configuration files. Instead, use secure vaults or environment variables for managing sensitive information.</div><div>Implement multi-factor authentication and strong password policies to enhance the security of user accounts and protect against unauthorized access.</div><div>Regularly rotate credentials and keys, especially after personnel changes or when a breach is suspected.</div></div></div></div></div></div>		

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High

Rec2. Use TLS for communications and protect stored data with encryption C-WEB-APPLICATION-SERVER-SIDE-CNT-05 Very high

Component: CLI Application

Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High

PCI-DSS-v3.2.1: 12.3.9

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

2 Recommended countermeasures

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High Not tested

- State: Recommended
- Description:

To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.

Implementation Steps:

 - Role-Based Access Control (RBAC):** Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.
 - Permission Validation:** For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.
 - Enforce Fine-Grained Access Control:** Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.
 - Session Management:** Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.
 - Audit Logs:** Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.
- References:
 - [OWASP Access Control Cheat Sheet](#)

Rec2. Use TLS for communications and protect stored data with encryption C-WEB-APPLICATION-SERVER-SIDE-CNT-05 Very high Not tested

- State: Recommended
- Description:

To secure data during transmission and at rest, ensure that all communications are protected using Transport Layer Security (TLS), and that stored data is encrypted using strong encryption algorithms. Additionally, implement robust key management practices to safeguard encryption keys and ensure that only authorized users and systems can access sensitive data.

Implementation Steps:

 - Enforce TLS for All Communications:** Configure all communication channels, including APIs, web traffic, and data transfers, to use TLS (preferably TLS 1.2 or higher) to protect data in transit.
 - Encrypt Stored Data:** Use strong encryption algorithms (e.g., AES-256) to encrypt sensitive data at rest, ensuring that unauthorized users cannot access or manipulate it.
 - Implement Key Management Practices:** Use a centralized key management system (KMS) to securely generate, store, and rotate encryption keys. Enforce access controls to ensure that only authorized systems can access encryption keys.
 - Regularly Update and Rotate Keys:** Set up automated key rotation policies to periodically update encryption keys and minimize the risk of key compromise.
- References:
 - [OWASP Transport Layer Security \(TLS\) Cheat Sheet](#)

Component: CLI Application

1 Recommended countermeasures

Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High Not tested

- State: Recommended
- Description:

To enhance the security of your CLI application, it's essential to focus on two pivotal areas: configuration management and credential management. A strategic approach to these components not only fortifies your application against unauthorized access but also ensures the safeguarding of sensitive information. Below, we detail actionable steps developers can take to implement these practices effectively:

 - Robust Configuration Management Process:**
 - Begin by establishing a standardized process for configuring systems and applications. This includes predefined security settings that align with industry best practices and compliance requirements.
 - Regularly review and update configurations to address newly identified vulnerabilities and security threats. Utilize automated tools where possible to ensure configurations remain consistent across environments.
 - Document all configuration changes and maintain version control to enable audit trails and rollback capabilities in case of issues.
 - Regular Security Reviews:**
 - Conduct periodic security reviews of system and application configurations to identify potential security gaps or misconfigurations.
 - Engage in penetration testing and vulnerability assessments to evaluate the effectiveness of current security configurations and identify areas for improvement.
 - Secure Credential Management:**
 - Store credentials (e.g., passwords, API keys, secret tokens) using encrypted storage solutions. Ensure encryption keys are managed securely and access to them is tightly controlled.
 - Avoid hard-coding credentials in source code or configuration files. Instead, use secure vaults or environment variables for managing sensitive information.
 - Implement multi-factor authentication and strong password policies to enhance the security of user accounts and protect against unauthorized access.
 - Regularly rotate credentials and keys, especially after personnel changes or when a breach is suspected.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High
Rec2. Use TLS for communications and protect stored data with encryption C-WEB-APPLICATION-SERVER-SIDE-CNT-05 Very high

Component: CLI Application

Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High

PCI-DSS-v3.2.1: 6.4.2

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

2 Recommended countermeasures

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High Not tested

- State: Recommended
- Description:

To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.

Implementation Steps:

 - Role-Based Access Control (RBAC):** Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.
 - Permission Validation:** For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.
 - Enforce Fine-Grained Access Control:** Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.
 - Session Management:** Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.
 - Audit Logs:** Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.

References:

 - [OWASP Access Control Cheat Sheet](#)

Rec2. Configure XML parsers to disallow external entities and DTD processing C-WEB-APPLICATION-SERVER-SIDE-CNT-10 Very high Not tested

- State: Recommended
- Description:

To prevent XML-based attacks such as XML External Entity (XXE) and Denial of Service (DoS) attacks, configure XML parsers to disallow external entities and Document Type Definition (DTD) processing by default. This ensures that any incoming XML documents cannot trigger external requests or load external data that could compromise the system, leak sensitive data, or cause resource exhaustion.

Implementation Steps:

 - Disable External Entity Resolution:** Configure XML parsers to explicitly disable the resolution of external entities by setting options or flags that prevent the parser from fetching external resources.
 - Disable DTD Processing:** Ensure that the XML parser does not process DTDs, which can be used to define and reference external entities or cause denial-of-service attacks via large or nested DTDs.
 - Use Secure Parsers:** Ensure that you are using updated and secure XML parsing libraries that offer built-in protections against XXE and other XML-based vulnerabilities.
 - Validate XML Input:** Before parsing, validate XML input to ensure it conforms to a known schema or structure, mitigating any unexpected or malicious data.

References:

 - [OWASP XML External Entity \(XXE\) Prevention Cheat Sheet](#)

Component: CLI Application

1 Recommended countermeasures

Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High Not tested

- State: Recommended
- Description:

To enhance the security of your CLI application, it's essential to focus on two pivotal areas: configuration management and credential management. A strategic approach to these components not only fortifies your application against unauthorized access but also ensures the safeguarding of sensitive information. Below, we detail actionable steps developers can take to implement these practices effectively:

 - Robust Configuration Management Process:**
 - Begin by establishing a standardized process for configuring systems and applications. This includes predefined security settings that align with industry best practices and compliance requirements.
 - Regularly review and update configurations to address newly identified vulnerabilities and security threats. Utilize automated tools where possible to ensure configurations remain consistent across environments.
 - Document all configuration changes and maintain version control to enable audit trails and rollback capabilities in case of issues.
 - Regular Security Reviews:**
 - Conduct periodic security reviews of system and application configurations to identify potential security gaps or misconfigurations.
 - Engage in penetration testing and vulnerability assessments to evaluate the effectiveness of current security configurations and identify areas for improvement.
 - Secure Credential Management:**
 - Store credentials (e.g., passwords, API keys, secret tokens) using encrypted storage solutions. Ensure encryption keys are managed securely and access to them is tightly controlled.
 - Avoid hard-coding credentials in source code or configuration files. Instead, use secure vaults or environment variables for managing sensitive information.
 - Implement multi-factor authentication and strong password policies to enhance the security of user accounts and protect against unauthorized access.
 - Regularly rotate credentials and keys, especially after personnel changes or when a breach is suspected.

Component: Source Code Repository

1 Recommended countermeasures

Rec1. Enforce a model of least privilege for source code repositories C-SOURCE-CODE-REPOSITORY-CNT-03 High Not tested

- State: Recommended
- Description:

Least privilege is the principle that a security architecture should be designed so that each entity is granted the minimum system resources and authorizations that the entity needs to perform its function.

Remediation:

Consider the exposure of your repository. Technically enforce a model of least privilege for who can make changes to your code repository. Make sure all activity is attributable. Additionally, access to the repository should be revoked swiftly when no longer required, or in the event of compromise

Reference:

 - [Protect access credentials](#)

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

- Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High
- Rec2. Configure XML parsers to disallow external entities and DTD processing C-WEB-APPLICATION-SERVER-SIDE-CNT-10 Very high

Component: CLI Application

- Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High

Component: Source Code Repository

- Rec1. Enforce a model of least privilege for source code repositories C-SOURCE-CODE-REPOSITORY-CNT-03 High

PCI-DSS-v3.2.1: 7.1.4

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: CLI Application

1 Recommended countermeasures

Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High Not tested

State: Recommended

Description:

To enhance the security of your CLI application, it's essential to focus on two pivotal areas: configuration management and credential management. A strategic approach to these components not only fortifies your application against unauthorized access but also ensures the safeguarding of sensitive information. Below, we detail actionable steps developers can take to implement these practices effectively:

- Robust Configuration Management Process:**
 - Begin by establishing a standardized process for configuring systems and applications. This includes predefined security settings that align with industry best practices and compliance requirements.
 - Regularly review and update configurations to address newly identified vulnerabilities and security threats. Utilize automated tools where possible to ensure configurations remain consistent across environments.
 - Document all configuration changes and maintain version control to enable audit trails and rollback capabilities in case of issues.
- Regular Security Reviews:**
 - Conduct periodic security reviews of system and application configurations to identify potential security gaps or misconfigurations.
 - Engage in penetration testing and vulnerability assessments to evaluate the effectiveness of current security configurations and identify areas for improvement.
- Secure Credential Management:**
 - Store credentials (e.g., passwords, API keys, secret tokens) using encrypted storage solutions. Ensure encryption keys are managed securely and access to them is tightly controlled.
 - Avoid hard-coding credentials in source code or configuration files. Instead, use secure vaults or environment variables for managing sensitive information.
 - Implement multi-factor authentication and strong password policies to enhance the security of user accounts and protect against unauthorized access.
 - Regularly rotate credentials and keys, especially after personnel changes or when a breach is suspected.

Component: Source Code Repository

1 Recommended countermeasures

Rec1. Protect access credentials in source code repositories C-SOURCE-CODE-REPOSITORY-CNT-02 Very high Not tested

State: Recommended

Description:

User access to repositories is often authenticated using credentials, such as passwords or private keys. Loss of these credentials may allow an attacker to gain unauthorized access.

Remediation:
Ensure that developers are encouraged to protect these credentials while they are used and managed within your development environment. Private keys should be password protected. Backing them onto a hardware token that makes them harder to abuse. Consider the option of rotating access keys.

Reference:

- [Protect access credentials](#)

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: CLI Application

- Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High

Component: Source Code Repository

- Rec1. Protect access credentials in source code repositories C-SOURCE-CODE-REPOSITORY-CNT-02 Very high

PCI-DSS-v3.2.1: 8.1.5

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

2 Recommended countermeasures

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High Not tested

- State: Recommended
- Description:

To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.
- Implementation Steps:
 - Role-Based Access Control (RBAC):** Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.
 - Permission Validation:** For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.
 - Enforce Fine-Grained Access Control:** Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.
 - Session Management:** Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.
 - Audit Logs:** Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.
- References:
 - [OWASP Access Control Cheat Sheet](#)

Rec2. Use TLS for communications and protect stored data with encryption C-WEB-APPLICATION-SERVER-SIDE-CNT-05 Very high Not tested

- State: Recommended
- Description:

To secure data during transmission and at rest, ensure that all communications are protected using Transport Layer Security (TLS), and that stored data is encrypted using strong encryption algorithms. Additionally, implement robust key management practices to safeguard encryption keys and ensure that only authorized users and systems can access sensitive data.
- Implementation Steps:
 - Enforce TLS for All Communications:** Configure all communication channels, including APIs, web traffic, and data transfers, to use TLS (preferably TLS 1.2 or higher) to protect data in transit.
 - Encrypt Stored Data:** Use strong encryption algorithms (e.g., AES-256) to encrypt sensitive data at rest, ensuring that unauthorized users cannot access or manipulate it.
 - Implement Key Management Practices:** Use a centralized key management system (KMS) to securely generate, store, and rotate encryption keys. Enforce access controls to ensure that only authorized systems can access encryption keys.
 - Regularly Update and Rotate Keys:** Set up automated key rotation policies to periodically update encryption keys and minimize the risk of key compromise.
- References:
 - [OWASP Transport Layer Security \(TLS\) Cheat Sheet](#)

Component: CLI Application

1 Recommended countermeasures

Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High Not tested

- State: Recommended
- Description:

To enhance the security of your CLI application, it's essential to focus on two pivotal areas: configuration management and credential management. A strategic approach to these components not only fortifies your application against unauthorized access but also ensures the safeguarding of sensitive information. Below, we detail actionable steps developers can take to implement these practices effectively:
- Robust Configuration Management Process:
 - Begin by establishing a standardized process for configuring systems and applications. This includes predefined security settings that align with industry best practices and compliance requirements.
 - Regularly review and update configurations to address newly identified vulnerabilities and security threats. Utilize automated tools where possible to ensure configurations remain consistent across environments.
 - Document all configuration changes and maintain version control to enable audit trails and rollback capabilities in case of issues.
- Regular Security Reviews:
 - Conduct periodic security reviews of system and application configurations to identify potential security gaps or misconfigurations.
 - Engage in penetration testing and vulnerability assessments to evaluate the effectiveness of current security configurations and identify areas for improvement.
- Secure Credential Management:
 - Store credentials (e.g., passwords, API keys, secret tokens) using encrypted storage solutions. Ensure encryption keys are managed securely and access to them is tightly controlled.
 - Avoid hard-coding credentials in source code or configuration files. Instead, use secure vaults or environment variables for managing sensitive information.
 - Implement multi-factor authentication and strong password policies to enhance the security of user accounts and protect against unauthorized access.
 - Regularly rotate credentials and keys, especially after personnel changes or when a breach is suspected.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High

Rec2. Use TLS for communications and protect stored data with encryption C-WEB-APPLICATION-SERVER-SIDE-CNT-05 Very high

Component: CLI Application

Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High

PCI-DSS-v3.2.1: 8.2.2

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: CLI Application

1 Recommended countermeasures

Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High Not tested

- State: Recommended
- Description:

To enhance the security of your CLI application, it's essential to focus on two pivotal areas: configuration management and credential management. A strategic approach to these components not only fortifies your application against unauthorized access but also ensures the safeguarding of sensitive information. Below, we detail actionable steps developers can take to implement these practices effectively:

- **Robust Configuration Management Process:**
 - Begin by establishing a standardized process for configuring systems and applications. This includes predefined security settings that align with industry best practices and compliance requirements.
 - Regularly review and update configurations to address newly identified vulnerabilities and security threats. Utilize automated tools where possible to ensure configurations remain consistent across environments.
 - Document all configuration changes and maintain version control to enable audit trails and rollback capabilities in case of issues.
- **Regular Security Reviews:**
 - Conduct periodic security reviews of system and application configurations to identify potential security gaps or misconfigurations.
 - Engage in penetration testing and vulnerability assessments to evaluate the effectiveness of current security configurations and identify areas for improvement.
- **Secure Credential Management:**
 - Store credentials (e.g., passwords, API keys, secret tokens) using encrypted storage solutions. Ensure encryption keys are managed securely and access to them is tightly controlled.
 - Avoid hard-coding credentials in source code or configuration files. Instead, use secure vaults or environment variables for managing sensitive information.
 - Implement multi-factor authentication and strong password policies to enhance the security of user accounts and protect against unauthorized access.
 - Regularly rotate credentials and keys, especially after personnel changes or when a breach is suspected.

Component: Source Code Repository

1 Recommended countermeasures

Rec1. Protect access credentials in source code repositories C-SOURCE-CODE-REPOSITORY-CNT-02 Very high Not tested

- **State:** Recommended
- **Description:**

User access to repositories is often authenticated using credentials, such as passwords or private keys. Loss of these credentials may allow an attacker to gain unauthorized access.

Remediation:

Ensure that developers are encouraged to protect these credentials while they are used and managed within your development environment.

Private keys should be password protected. Backing them onto a hardware token that makes them harder to abuse.

Consider the option of rotating access keys.

Reference:

 - [Protect access credentials](#)

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: CLI Application

Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High

Component: Source Code Repository

Rec1. Protect access credentials in source code repositories C-SOURCE-CODE-REPOSITORY-CNT-02 Very high

PCI-DSS-v3.2.1: 8.5.1

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

2 Recommended countermeasures

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High Not tested

- **State:** Recommended
- **Description:**

To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.

Implementation Steps:

 1. **Role-Based Access Control (RBAC):** Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.
 2. **Permission Validation:** For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.
 3. **Enforce Fine-Grained Access Control:** Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.
 4. **Session Management:** Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.
 5. **Audit Logs:** Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.

References:

 - [OWASP Access Control Cheat Sheet](#)

Rec2. Use TLS for communications and protect stored data with encryption C-WEB-APPLICATION-SERVER-SIDE-CNT-05 Very high Not tested

- **State:** Recommended
- **Description:**

To secure data during transmission and at rest, ensure that all communications are protected using Transport Layer Security (TLS), and that stored data is encrypted using strong encryption algorithms. Additionally, implement robust key management practices to safeguard encryption keys and ensure that only authorized users and systems can access sensitive data.

Implementation Steps:

 1. **Enforce TLS for All Communications:** Configure all communication channels, including APIs, web traffic, and data transfers, to use TLS (preferably TLS 1.2 or higher) to protect data in transit.
 2. **Encrypt Stored Data:** Use strong encryption algorithms (e.g., AES-256) to encrypt sensitive data at rest, ensuring that unauthorized users cannot access or manipulate it.
 3. **Implement Key Management Practices:** Use a centralized key management system (KMS) to securely generate, store, and rotate encryption keys. Enforce access controls to ensure that only authorized systems can access encryption keys.
 4. **Regularly Update and Rotate Keys:** Set up automated key rotation policies to periodically update encryption keys and minimize the risk of key compromise.

References:

- [OWASP Transport Layer Security \(TLS\) Cheat Sheet](#)

Component: CLI Application

1 Recommended countermeasures

Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High Not tested

- State: Recommended
- Description:

To enhance the security of your CLI application, it's essential to focus on two pivotal areas: configuration management and credential management. A strategic approach to these components not only fortifies your application against unauthorized access but also ensures the safeguarding of sensitive information. Below, we detail actionable steps developers can take to implement these practices effectively:

 - **Robust Configuration Management Process:**
 - Begin by establishing a standardized process for configuring systems and applications. This includes predefined security settings that align with industry best practices and compliance requirements.
 - Regularly review and update configurations to address newly identified vulnerabilities and security threats. Utilize automated tools where possible to ensure configurations remain consistent across environments.
 - Document all configuration changes and maintain version control to enable audit trails and rollback capabilities in case of issues.
 - **Regular Security Reviews:**
 - Conduct periodic security reviews of system and application configurations to identify potential security gaps or misconfigurations.
 - Engage in penetration testing and vulnerability assessments to evaluate the effectiveness of current security configurations and identify areas for improvement.
 - **Secure Credential Management:**
 - Store credentials (e.g., passwords, API keys, secret tokens) using encrypted storage solutions. Ensure encryption keys are managed securely and access to them is tightly controlled.
 - Avoid hard-coding credentials in source code or configuration files. Instead, use secure vaults or environment variables for managing sensitive information.
 - Implement multi-factor authentication and strong password policies to enhance the security of user accounts and protect against unauthorized access.
 - Regularly rotate credentials and keys, especially after personnel changes or when a breach is suspected.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High
Rec2. Use TLS for communications and protect stored data with encryption C-WEB-APPLICATION-SERVER-SIDE-CNT-05 Very high

Component: CLI Application

Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High

PCI-DSS-v3.2.1: 2.2

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

4 Recommended countermeasures

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High Not tested

- State: Recommended
- Description:

To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.

Implementation Steps:

 1. **Role-Based Access Control (RBAC):** Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.
 2. **Permission Validation:** For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.
 3. **Enforce Fine-Grained Access Control:** Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.
 4. **Session Management:** Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.
 5. **Audit Logs:** Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.

References:

 - [OWASP Access Control Cheat Sheet](#)

Rec2. Configure XML parsers to disallow external entities and DTD processing C-WEB-APPLICATION-SERVER-SIDE-CNT-10 Very high Not tested

- State: Recommended
- Description:

To prevent XML-based attacks such as XML External Entity (XXE) and Denial of Service (DoS) attacks, configure XML parsers to disallow external entities and Document Type Definition (DTD) processing by default. This ensures that any incoming XML documents cannot trigger external requests or load external data that could compromise the system, leak sensitive data, or cause resource exhaustion.

Implementation Steps:

 1. **Disable External Entity Resolution:** Configure XML parsers to explicitly disable the resolution of external entities by setting options or flags that prevent the parser from fetching external resources.
 2. **Disable DTD Processing:** Ensure that the XML parser does not process DTDs, which can be used to define and reference external entities or cause denial-of-service attacks via large or nested DTDs.
 3. **Use Secure Parsers:** Ensure that you are using updated and secure XML parsing libraries that offer built-in protections against XXE and other XML-based vulnerabilities.
 4. **Validate XML Input:** Before parsing, validate XML input to ensure it conforms to a known schema or structure, mitigating any unexpected or malicious data.

References:

 - [OWASP XML External Entity \(XXE\) Prevention Cheat Sheet](#)

Rec3. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High ☐ Not tested

- State: Recommended
- Description:

To minimize the attack surface and enhance security, disable any unused services, remove default accounts, and apply security updates promptly. By doing so, you reduce the risk of exploitation through unneeded services or default credentials and ensure that the system is protected from known vulnerabilities. Additionally, use secure configurations to prevent unauthorized access and ensure that the system is optimally protected.

Implementation Steps:

 - Disable Unused Services:** Identify and disable any services that are not needed for the application or system to function. This limits the number of potential attack vectors.
 - Remove Default Accounts:** Remove or disable default accounts and ensure that all active accounts are assigned strong, unique credentials. Avoid using default settings that might be easily guessed by attackers.
 - Apply Security Updates:** Regularly check for and apply security patches to all systems and applications. Set up an automated patch management process to ensure that critical updates are applied as soon as they are released.
 - Use Secure Configurations:** Ensure that all configurations, such as database, server, and application settings, follow security best practices, including strong encryption, least privilege access, and secure communication protocols.

References:

 - [OWASP Secure Configuration Guide](#)

Rec4. Block or validate all outbound requests C-WEB-APPLICATION-SERVER-SIDE-CNT-11 Medium ☐ Not tested

- State: Recommended
- Description:

To reduce the risk of data exfiltration or unauthorized system communication, block or carefully validate all outbound requests made by the application. Ensure that only requests to specific, trusted domains and IP addresses are allowed, and that any outbound request to unapproved destinations is blocked or flagged for further review. This control prevents the application from communicating with malicious external systems and protects sensitive data from being leaked.

Implementation Steps:

 - Implement Outbound Request Filtering:** Use a web application firewall (WAF) or proxy to block or monitor any outbound requests that are not to trusted domains or IP addresses.
 - Validate Outbound Requests:** Before allowing outbound communication, validate the destination domain and IP address to ensure they are within an approved list of resources necessary for the application.
 - Set Domain and IP Allowlists:** Define an allowlist of specific domains and IP addresses that the application is authorized to interact with, and block all other outbound traffic.
 - Monitor and Log Outbound Requests:** Continuously monitor and log all outbound requests for suspicious activity or attempts to communicate with unapproved destinations.
 - Review and Update Allowlist Regularly:** Periodically review the allowlist to ensure that it remains up-to-date with the application's legitimate requirements.

References:

 - [Security and Privacy Controls for Information Systems and Organizations](#)

Component: CLI Application

1 Recommended countermeasures

Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High ☐ Not tested

- State: Recommended
- Description:

To enhance the security of your CLI application, it's essential to focus on two pivotal areas: configuration management and credential management. A strategic approach to these components not only fortifies your application against unauthorized access but also ensures the safeguarding of sensitive information. Below, we detail actionable steps developers can take to implement these practices effectively:

 - Robust Configuration Management Process:**
 - Begin by establishing a standardized process for configuring systems and applications. This includes predefined security settings that align with industry best practices and compliance requirements.
 - Regularly review and update configurations to address newly identified vulnerabilities and security threats. Utilize automated tools where possible to ensure configurations remain consistent across environments.
 - Document all configuration changes and maintain version control to enable audit trails and rollback capabilities in case of issues.
 - Regular Security Reviews:**
 - Conduct periodic security reviews of system and application configurations to identify potential security gaps or misconfigurations.
 - Engage in penetration testing and vulnerability assessments to evaluate the effectiveness of current security configurations and identify areas for improvement.
 - Secure Credential Management:**
 - Store credentials (e.g., passwords, API keys, secret tokens) using encrypted storage solutions. Ensure encryption keys are managed securely and access to them is tightly controlled.
 - Avoid hard-coding credentials in source code or configuration files. Instead, use secure vaults or environment variables for managing sensitive information.
 - Implement multi-factor authentication and strong password policies to enhance the security of user accounts and protect against unauthorized access.
 - Regularly rotate credentials and keys, especially after personnel changes or when a breach is suspected.

Component: Document-Oriented Database

1 Recommended countermeasures

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CM5 Very high ☐ Not tested

- State: Recommended
- Description:

Conducting regular configuration audits of a Document-Oriented Database is vital for ensuring that security settings and configurations are aligned with best practices and organizational security policies. Here's how to systematically approach these audits:

 - Establish a Baseline Configuration
 - Identify Security Settings:** Document the optimal security configurations for your database environment. This includes access controls, encryption settings, network configurations, and any specific database security features.
 - Create a Configuration Template:** Develop a template or checklist that outlines all security-related configuration settings. This template will serve as the baseline for regular audits.
 - Automate the Audit Process
 - Use Configuration Management Tools:** Implement tools that can automate the process of detecting deviations from the baseline configuration. Tools like Puppet, Chef, Ansible, or database-specific management solutions can help.
 - Schedule Regular Scans:** Configure these tools to perform regular scans of your document-oriented database environment to identify and report any configuration changes or non-compliance with the baseline.
 - Review Access Controls
 - Validate User Access:** Regularly review user accounts, roles, and permissions to ensure that they adhere to the principle of least privilege. Remove or adjust any excessive permissions.
 - Audit Authentication Mechanisms:** Verify that authentication methods (password policies, multi-factor authentication, etc.) are properly configured and enforced.
 - Verify Network Configurations
 - Check Network Access Controls:** Ensure that the database is not exposed to unauthorized networks. Validate firewall rules and other network security configurations that control access to the database.
 - Secure Communication Channels:** Confirm that data in transit is encrypted and that secure communication protocols are in use.
 - Confirm Data Protection Measures
 - Data Encryption:** Verify that data at rest is encrypted according to best practices. Check the configuration of encryption mechanisms and the management of encryption keys.
 - Backup and Recovery:** Review the configuration of backup processes to ensure that data is securely backed up and that encryption is maintained for backups.
 - Document and Resolve Findings
 - Report Findings:** Document any deviations from the baseline configuration identified during the audit. Classify findings based on their potential impact on security.
 - Plan Remediation:** Develop a plan for addressing high-priority findings. This may include reconfiguring settings, updating software, or modifying access controls.

- **Track Progress:** Maintain records of audits, findings, and remediation efforts to track improvements over time and to provide accountability.
7. Update Baseline Configurations
- **Incorporate Changes:** As you make changes to your database environment or as new security best practices emerge, update your baseline configuration and templates accordingly.
 - **Review Periodically:** Regularly review the baseline configuration to ensure it remains relevant and effective in mitigating security risks.
8. Educate and Train
- **Team Awareness:** Ensure that your team is aware of the configuration audit process, the importance of adhering to the baseline configurations, and the role they play in maintaining database security.
 - **Continuous Learning:** Stay informed about new security features offered by your document-oriented database system and incorporate relevant features into your security baseline.
- References and Good Security Practices**
- **Industry Best Practices:** Follow guidelines and security checklists provided by the database vendor, as well as security standards relevant to your industry (e.g., CIS Benchmarks, NIST Guidelines).
 - **Security Communities:** Engage with security communities and forums related to your database technology to stay updated on the latest security advisories and recommendations.
- Regular configuration audits are a proactive step in maintaining the security and integrity of your document-oriented database, helping to prevent unauthorized access, data leaks, and ensuring compliance with data protection regulations.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

- Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High
- Rec2. Configure XML parsers to disallow external entities and DTD processing C-WEB-APPLICATION-SERVER-SIDE-CNT-10 Very high
- Rec3. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High
- Rec4. Block or validate all outbound requests C-WEB-APPLICATION-SERVER-SIDE-CNT-11 Medium

Component: CLI Application

- Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High

Component: Document-Oriented Database

- Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CM5 Very high

PCI-DSS-v3.2.1: 2.3

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

2 Recommended countermeasures

- Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High Not tested
- State: Recommended
- Description:
- To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.
- Implementation Steps:
1. **Role-Based Access Control (RBAC):** Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.
 2. **Permission Validation:** For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.
 3. **Enforce Fine-Grained Access Control:** Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.
 4. **Session Management:** Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.
 5. **Audit Logs:** Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.
- References:
- [OWASP Access Control Cheat Sheet](#)
- Rec2. Use TLS for communications and protect stored data with encryption C-WEB-APPLICATION-SERVER-SIDE-CNT-05 Very high Not tested
- State: Recommended
- Description:
- To secure data during transmission and at rest, ensure that all communications are protected using Transport Layer Security (TLS), and that stored data is encrypted using strong encryption algorithms. Additionally, implement robust key management practices to safeguard encryption keys and ensure that only authorized users and systems can access sensitive data.
- Implementation Steps:
1. **Enforce TLS for All Communications:** Configure all communication channels, including APIs, web traffic, and data transfers, to use TLS (preferably TLS 1.2 or higher) to protect data in transit.
 2. **Encrypt Stored Data:** Use strong encryption algorithms (e.g., AES-256) to encrypt sensitive data at rest, ensuring that unauthorized users cannot access or manipulate it.
 3. **Implement Key Management Practices:** Use a centralized key management system (KMS) to securely generate, store, and rotate encryption keys. Enforce access controls to ensure that only authorized systems can access encryption keys.
 4. **Regularly Update and Rotate Keys:** Set up automated key rotation policies to periodically update encryption keys and minimize the risk of key compromise.
- References:
- [OWASP Transport Layer Security \(TLS\) Cheat Sheet](#)

Component: CLI Application

1 Recommended countermeasures

- Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High Not tested
- State: Recommended
- Description:

To enhance the security of your CLI application, it's essential to focus on two pivotal areas: configuration management and credential management. A strategic approach to these components not only fortifies your application against unauthorized access but also ensures the safeguarding of sensitive information. Below, we detail actionable steps developers can take to implement these practices effectively:

- **Robust Configuration Management Process:**
 - Begin by establishing a standardized process for configuring systems and applications. This includes predefined security settings that align with industry best practices and compliance requirements.
 - Regularly review and update configurations to address newly identified vulnerabilities and security threats. Utilize automated tools where possible to ensure configurations remain consistent across environments.
 - Document all configuration changes and maintain version control to enable audit trails and rollback capabilities in case of issues.
- **Regular Security Reviews:**
 - Conduct periodic security reviews of system and application configurations to identify potential security gaps or misconfigurations.
 - Engage in penetration testing and vulnerability assessments to evaluate the effectiveness of current security configurations and identify areas for improvement.
- **Secure Credential Management:**
 - Store credentials (e.g., passwords, API keys, secret tokens) using encrypted storage solutions. Ensure encryption keys are managed securely and access to them is tightly controlled.
 - Avoid hard-coding credentials in source code or configuration files. Instead, use secure vaults or environment variables for managing sensitive information.
 - Implement multi-factor authentication and strong password policies to enhance the security of user accounts and protect against unauthorized access.
 - Regularly rotate credentials and keys, especially after personnel changes or when a breach is suspected.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

- Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High
- Rec2. Use TLS for communications and protect stored data with encryption C-WEB-APPLICATION-SERVER-SIDE-CNT-05 Very high

Component: CLI Application

- Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High

PCI-DSS-v3.2.1: 7.1

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

2 Recommended countermeasures

- Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High Not tested
- State: Recommended
 - Description:

To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.
 - Implementation Steps:
 1. **Role-Based Access Control (RBAC):** Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.
 2. **Permission Validation:** For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.
 3. **Enforce Fine-Grained Access Control:** Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.
 4. **Session Management:** Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.
 5. **Audit Logs:** Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.
 - References:
 - [OWASP Access Control Cheat Sheet](#)
- Rec2. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High Not tested
- State: Recommended
 - Description:

To minimize the attack surface and enhance security, disable any unused services, remove default accounts, and apply security updates promptly. By doing so, you reduce the risk of exploitation through unneeded services or default credentials and ensure that the system is protected from known vulnerabilities. Additionally, use secure configurations to prevent unauthorized access and ensure that the system is optimally protected.
 - Implementation Steps:
 1. **Disable Unused Services:** Identify and disable any services that are not needed for the application or system to function. This limits the number of potential attack vectors.
 2. **Remove Default Accounts:** Remove or disable default accounts and ensure that all active accounts are assigned strong, unique credentials. Avoid using default settings that might be easily guessed by attackers.
 3. **Apply Security Updates:** Regularly check for and apply security patches to all systems and applications. Set up an automated patch management process to ensure that critical updates are applied as soon as they are released.
 4. **Use Secure Configurations:** Ensure that all configurations, such as database, server, and application settings, follow security best practices, including strong encryption, least privilege access, and secure communication protocols.
 - References:
 - [OWASP Secure Configuration Guide](#)

Component: CLI Application

1 Recommended countermeasures

- Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High Not tested
- State: Recommended
 - Description:

To enhance the security of your CLI application, it's essential to focus on two pivotal areas: configuration management and credential management. A strategic approach to these components not only fortifies your application against unauthorized access but also ensures the safeguarding of sensitive information. Below, we detail actionable steps developers can take to implement these

practices effectively:

- **Robust Configuration Management Process:**
 - Begin by establishing a standardized process for configuring systems and applications. This includes predefined security settings that align with industry best practices and compliance requirements.
 - Regularly review and update configurations to address newly identified vulnerabilities and security threats. Utilize automated tools where possible to ensure configurations remain consistent across environments.
 - Document all configuration changes and maintain version control to enable audit trails and rollback capabilities in case of issues.
- **Regular Security Reviews:**
 - Conduct periodic security reviews of system and application configurations to identify potential security gaps or misconfigurations.
 - Engage in penetration testing and vulnerability assessments to evaluate the effectiveness of current security configurations and identify areas for improvement.
- **Secure Credential Management:**
 - Store credentials (e.g., passwords, API keys, secret tokens) using encrypted storage solutions. Ensure encryption keys are managed securely and access to them is tightly controlled.
 - Avoid hard-coding credentials in source code or configuration files. Instead, use secure vaults or environment variables for managing sensitive information.
 - Implement multi-factor authentication and strong password policies to enhance the security of user accounts and protect against unauthorized access.
 - Regularly rotate credentials and keys, especially after personnel changes or when a breach is suspected.

Component: Document-Oriented Database

1 Recommended countermeasures

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CMS Very high Not tested

- **State:** Recommended
- **Description:**

Conducting regular configuration audits of a Document-Oriented Database is vital for ensuring that security settings and configurations are aligned with best practices and organizational security policies. Here's how to systematically approach these audits:

 1. Establish a Baseline Configuration
 - **Identify Security Settings:** Document the optimal security configurations for your database environment. This includes access controls, encryption settings, network configurations, and any specific database security features.
 - **Create a Configuration Template:** Develop a template or checklist that outlines all security-related configuration settings. This template will serve as the baseline for regular audits.
 2. Automate the Audit Process
 - **Use Configuration Management Tools:** Implement tools that can automate the process of detecting deviations from the baseline configuration. Tools like Puppet, Chef, Ansible, or database-specific management solutions can help.
 - **Schedule Regular Scans:** Configure these tools to perform regular scans of your document-oriented database environment to identify and report any configuration changes or non-compliance with the baseline.
 3. Review Access Controls
 - **Validate User Access:** Regularly review user accounts, roles, and permissions to ensure that they adhere to the principle of least privilege. Remove or adjust any excessive permissions.
 - **Audit Authentication Mechanisms:** Verify that authentication methods (password policies, multi-factor authentication, etc.) are properly configured and enforced.
 4. Verify Network Configurations
 - **Check Network Access Controls:** Ensure that the database is not exposed to unauthorized networks. Validate firewall rules and other network security configurations that control access to the database.
 - **Secure Communication Channels:** Confirm that data in transit is encrypted and that secure communication protocols are in use.
 5. Confirm Data Protection Measures
 - **Data Encryption:** Verify that data at rest is encrypted according to best practices. Check the configuration of encryption mechanisms and the management of encryption keys.
 - **Backup and Recovery:** Review the configuration of backup processes to ensure that data is securely backed up and that encryption is maintained for backups.
 6. Document and Resolve Findings
 - **Report Findings:** Document any deviations from the baseline configuration identified during the audit. Classify findings based on their potential impact on security.
 - **Plan Remediation:** Develop a plan for addressing high-priority findings. This may include reconfiguring settings, updating software, or modifying access controls.
 - **Track Progress:** Maintain records of audits, findings, and remediation efforts to track improvements over time and to provide accountability.
 7. Update Baseline Configurations
 - **Incorporate Changes:** As you make changes to your database environment or as new security best practices emerge, update your baseline configuration and templates accordingly.
 - **Review Periodically:** Regularly review the baseline configuration to ensure it remains relevant and effective in mitigating security risks.
 8. Educate and Train
 - **Team Awareness:** Ensure that your team is aware of the configuration audit process, the importance of adhering to the baseline configurations, and the role they play in maintaining database security.
 - **Continuous Learning:** Stay informed about new security features offered by your document-oriented database system and incorporate relevant features into your security baseline.

References and Good Security Practices

 - **Industry Best Practices:** Follow guidelines and security checklists provided by the database vendor, as well as security standards relevant to your industry (e.g., CIS Benchmarks, NIST Guidelines).
 - **Security Communities:** Engage with security communities and forums related to your database technology to stay updated on the latest security advisories and recommendations.

Regular configuration audits are a proactive step in maintaining the security and integrity of your document-oriented database, helping to prevent unauthorized access, data leaks, and ensuring compliance with data protection regulations.

Component: Source Code Repository

1 Recommended countermeasures

Rec1. Enforce a model of least privilege for source code repositories C-SOURCE-CODE-REPOSITORY-CNT-03 High Not tested

- **State:** Recommended
- **Description:**

Least privilege is the principle that a security architecture should be designed so that each entity is granted the minimum system resources and authorizations that the entity needs to perform its function.

Remediation:

Consider the exposure of your repository. Technically enforce a model of least privilege for who can make changes to your code repository. Make sure all activity is attributable. Additionally, access to the repository should be revoked swiftly when no longer required, or in the event of compromise

Reference:

 - [Protect access credentials](#)

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

- Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High
- Rec2. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High

Component: CLI Application

- Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High

Component: Document-Oriented Database

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CM5 Very high

Component: Source Code Repository

Rec1. Enforce a model of least privilege for source code repositories C-SOURCE-CODE-REPOSITORY-CNT-03 High

PCI-DSS-v3.2.1: 7.2

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

2 Recommended countermeasures

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High Not tested

- State: Recommended
- Description:

To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.

Implementation Steps:

 - Role-Based Access Control (RBAC):** Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.
 - Permission Validation:** For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.
 - Enforce Fine-Grained Access Control:** Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.
 - Session Management:** Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.
 - Audit Logs:** Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.

References:

 - [OWASP Access Control Cheat Sheet](#)

Rec2. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High Not tested

- State: Recommended
- Description:

To minimize the attack surface and enhance security, disable any unused services, remove default accounts, and apply security updates promptly. By doing so, you reduce the risk of exploitation through unneeded services or default credentials and ensure that the system is protected from known vulnerabilities. Additionally, use secure configurations to prevent unauthorized access and ensure that the system is optimally protected.

Implementation Steps:

 - Disable Unused Services:** Identify and disable any services that are not needed for the application or system to function. This limits the number of potential attack vectors.
 - Remove Default Accounts:** Remove or disable default accounts and ensure that all active accounts are assigned strong, unique credentials. Avoid using default settings that might be easily guessed by attackers.
 - Apply Security Updates:** Regularly check for and apply security patches to all systems and applications. Set up an automated patch management process to ensure that critical updates are applied as soon as they are released.
 - Use Secure Configurations:** Ensure that all configurations, such as database, server, and application settings, follow security best practices, including strong encryption, least privilege access, and secure communication protocols.

References:

 - [OWASP Secure Configuration Guide](#)

Component: CLI Application

1 Recommended countermeasures

Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High Not tested

- State: Recommended
- Description:

To enhance the security of your CLI application, it's essential to focus on two pivotal areas: configuration management and credential management. A strategic approach to these components not only fortifies your application against unauthorized access but also ensures the safeguarding of sensitive information. Below, we detail actionable steps developers can take to implement these practices effectively:

 - Robust Configuration Management Process:**
 - Begin by establishing a standardized process for configuring systems and applications. This includes predefined security settings that align with industry best practices and compliance requirements.
 - Regularly review and update configurations to address newly identified vulnerabilities and security threats. Utilize automated tools where possible to ensure configurations remain consistent across environments.
 - Document all configuration changes and maintain version control to enable audit trails and rollback capabilities in case of issues.
 - Regular Security Reviews:**
 - Conduct periodic security reviews of system and application configurations to identify potential security gaps or misconfigurations.
 - Engage in penetration testing and vulnerability assessments to evaluate the effectiveness of current security configurations and identify areas for improvement.
 - Secure Credential Management:**
 - Store credentials (e.g., passwords, API keys, secret tokens) using encrypted storage solutions. Ensure encryption keys are managed securely and access to them is tightly controlled.
 - Avoid hard-coding credentials in source code or configuration files. Instead, use secure vaults or environment variables for managing sensitive information.
 - Implement multi-factor authentication and strong password policies to enhance the security of user accounts and protect against unauthorized access.
 - Regularly rotate credentials and keys, especially after personnel changes or when a breach is suspected.

Component: Document-Oriented Database

1 Recommended countermeasures

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CM5 Very high Not tested

- State: Recommended
- Description:

Conducting regular configuration audits of a Document-Oriented Database is vital for ensuring that security settings and configurations are aligned with best practices and organizational security policies. Here's how to systematically approach these audits:

1. Establish a Baseline Configuration
 - **Identify Security Settings:** Document the optimal security configurations for your database environment. This includes access controls, encryption settings, network configurations, and any specific database security features.
 - **Create a Configuration Template:** Develop a template or checklist that outlines all security-related configuration settings. This template will serve as the baseline for regular audits.
2. Automate the Audit Process
 - **Use Configuration Management Tools:** Implement tools that can automate the process of detecting deviations from the baseline configuration. Tools like Puppet, Chef, Ansible, or database-specific management solutions can help.
 - **Schedule Regular Scans:** Configure these tools to perform regular scans of your document-oriented database environment to identify and report any configuration changes or non-compliance with the baseline.
3. Review Access Controls
 - **Validate User Access:** Regularly review user accounts, roles, and permissions to ensure that they adhere to the principle of least privilege. Remove or adjust any excessive permissions.
 - **Audit Authentication Mechanisms:** Verify that authentication methods (password policies, multi-factor authentication, etc.) are properly configured and enforced.
4. Verify Network Configurations
 - **Check Network Access Controls:** Ensure that the database is not exposed to unauthorized networks. Validate firewall rules and other network security configurations that control access to the database.
 - **Secure Communication Channels:** Confirm that data in transit is encrypted and that secure communication protocols are in use.
5. Confirm Data Protection Measures
 - **Data Encryption:** Verify that data at rest is encrypted according to best practices. Check the configuration of encryption mechanisms and the management of encryption keys.
 - **Backup and Recovery:** Review the configuration of backup processes to ensure that data is securely backed up and that encryption is maintained for backups.
6. Document and Resolve Findings
 - **Report Findings:** Document any deviations from the baseline configuration identified during the audit. Classify findings based on their potential impact on security.
 - **Plan Remediation:** Develop a plan for addressing high-priority findings. This may include reconfiguring settings, updating software, or modifying access controls.
 - **Track Progress:** Maintain records of audits, findings, and remediation efforts to track improvements over time and to provide accountability.
7. Update Baseline Configurations
 - **Incorporate Changes:** As you make changes to your database environment or as new security best practices emerge, update your baseline configuration and templates accordingly.
 - **Review Periodically:** Regularly review the baseline configuration to ensure it remains relevant and effective in mitigating security risks.
8. Educate and Train
 - **Team Awareness:** Ensure that your team is aware of the configuration audit process, the importance of adhering to the baseline configurations, and the role they play in maintaining database security.
 - **Continuous Learning:** Stay informed about new security features offered by your document-oriented database system and incorporate relevant features into your security baseline.
- References and Good Security Practices
 - **Industry Best Practices:** Follow guidelines and security checklists provided by the database vendor, as well as security standards relevant to your industry (e.g., CIS Benchmarks, NIST Guidelines).
 - **Security Communities:** Engage with security communities and forums related to your database technology to stay updated on the latest security advisories and recommendations.

Regular configuration audits are a proactive step in maintaining the security and integrity of your document-oriented database, helping to prevent unauthorized access, data leaks, and ensuring compliance with data protection regulations.
- Component: Source Code Repository
- 1 Recommended countermeasures
- Rec1. Enforce a model of least privilege for source code repositories C-SOURCE-CODE-REPOSITORY-CNT-03 High Not tested

State: Recommended

Description:

Least privilege is the principle that a security architecture should be designed so that each entity is granted the minimum system resources and authorizations that the entity needs to perform its function.

Remediation:
Consider the exposure of your repository. Technically enforce a model of least privilege for who can make changes to your code repository. Make sure all activity is attributable. Additionally, access to the repository should be revoked swiftly when no longer required, or in the event of compromise

Reference:

 - [Protect access credentials](#)
- Recommended countermeasures
- Below are the recommended countermeasures ("Rec") by component and threat for standard reference.
- Component: Web Application - Server side
- Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High

Rec2. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High
- Component: CLI Application
- Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High
- Component: Document-Oriented Database
- Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CMS Very high
- Component: Source Code Repository
- Rec1. Enforce a model of least privilege for source code repositories C-SOURCE-CODE-REPOSITORY-CNT-03 High
- PCI-DSS-v3.2.1: 8.1
- Non-compliant countermeasures
- "Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.
- Component: Web Application - Server side
- 1 Recommended countermeasures
- Rec1. Enforce MFA, use secure cookies, and invalidate sessions C-WEB-APPLICATION-SERVER-SIDE-CNT-02 High Not tested

State: Recommended

Description:

To enhance authentication security, enforce multi-factor authentication (MFA) for all users, particularly for accessing sensitive resources. Use secure cookies to store session information and ensure they are configured with proper flags (e.g., HttpOnly, Secure) to prevent unauthorized access. Additionally, implement session invalidation after logout or inactivity to minimize the risk of session
- Report: CodeSentinel Thread Modeling

Compliance report - 2025-03-14T18:54:57.395420405Z

61 of 97

hijacking or unauthorized access.

Implementation Steps:

1. **Enforce MFA:** Configure MFA for all user accounts, requiring an additional verification step (e.g., SMS, authenticator app) beyond just username and password. This should be applied to all sensitive operations and accounts with high privileges.
2. **Use Secure Cookies:** Set cookies with the Secure flag (to ensure they are only sent over HTTPS), HttpOnly flag (to prevent access via JavaScript), and SameSite flag (to restrict cross-site request behavior), ensuring session data is protected.
3. **Invalidate Sessions on Logout or Inactivity:** Implement session expiration or timeouts for inactivity and invalidate user sessions upon logout to prevent session hijacking. Ensure that tokens are revoked immediately when no longer needed.
4. **Monitor Session Activity:** Continuously monitor and review session activity to detect any unauthorized access attempts or suspicious session behavior.

References:

- [OWASP Authentication Cheat Sheet](#)

Component: CLI Application

1 Recommended countermeasures

Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High Not tested

- State: Recommended
- Description:

To enhance the security of your CLI application, it's essential to focus on two pivotal areas: configuration management and credential management. A strategic approach to these components not only fortifies your application against unauthorized access but also ensures the safeguarding of sensitive information. Below, we detail actionable steps developers can take to implement these practices effectively:

 - **Robust Configuration Management Process:**
 - Begin by establishing a standardized process for configuring systems and applications. This includes predefined security settings that align with industry best practices and compliance requirements.
 - Regularly review and update configurations to address newly identified vulnerabilities and security threats. Utilize automated tools where possible to ensure configurations remain consistent across environments.
 - Document all configuration changes and maintain version control to enable audit trails and rollback capabilities in case of issues.
 - **Regular Security Reviews:**
 - Conduct periodic security reviews of system and application configurations to identify potential security gaps or misconfigurations.
 - Engage in penetration testing and vulnerability assessments to evaluate the effectiveness of current security configurations and identify areas for improvement.
 - **Secure Credential Management:**
 - Store credentials (e.g., passwords, API keys, secret tokens) using encrypted storage solutions. Ensure encryption keys are managed securely and access to them is tightly controlled.
 - Avoid hard-coding credentials in source code or configuration files. Instead, use secure vaults or environment variables for managing sensitive information.
 - Implement multi-factor authentication and strong password policies to enhance the security of user accounts and protect against unauthorized access.
 - Regularly rotate credentials and keys, especially after personnel changes or when a breach is suspected.

Component: Source Code Repository

1 Recommended countermeasures

Rec1. Protect access credentials in source code repositories C-SOURCE-CODE-REPOSITORY-CNT-02 Very high Not tested

- State: Recommended
- Description:

User access to repositories is often authenticated using credentials, such as passwords or private keys. Loss of these credentials may allow an attacker to gain unauthorized access.

Remediation:
Ensure that developers are encouraged to protect these credentials while they are used and managed within your development environment.
Private keys should be password protected. Backing them onto a hardware token that makes them harder to abuse.
Consider the option of rotating access keys.

Reference:

 - [Protect access credentials](#)

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Enforce MFA, use secure cookies, and invalidate sessions C-WEB-APPLICATION-SERVER-SIDE-CNT-02 High

Component: CLI Application

Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High

Component: Source Code Repository

Rec1. Protect access credentials in source code repositories C-SOURCE-CODE-REPOSITORY-CNT-02 Very high

PCI-DSS-v3.2.1: 8.2

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

1 Recommended countermeasures

Rec1. Enforce MFA, use secure cookies, and invalidate sessions C-WEB-APPLICATION-SERVER-SIDE-CNT-02 High Not tested

- State: Recommended
- Description:

To enhance authentication security, enforce multi-factor authentication (MFA) for all users, particularly for accessing sensitive resources. Use secure cookies to store session information and ensure they are configured with proper flags (e.g., HttpOnly, Secure) to prevent unauthorized access. Additionally, implement session invalidation after logout or inactivity to minimize the risk of session hijacking or unauthorized access.

Implementation Steps:

- 1. **Enforce MFA:** Configure MFA for all user accounts, requiring an additional verification step (e.g., SMS, authenticator app) beyond just username and password. This should be applied to all sensitive operations and accounts with high privileges.
- 2. **Use Secure Cookies:** Set cookies with the Secure flag (to ensure they are only sent over HTTPS), HttpOnly flag (to prevent access via JavaScript), and SameSite flag (to restrict cross-site request behavior), ensuring session data is protected.
- 3. **Invalidate Sessions on Logout or Inactivity:** Implement session expiration or timeouts for inactivity and invalidate user sessions upon logout to prevent session hijacking. Ensure that tokens are revoked immediately when no longer needed.
- 4. **Monitor Session Activity:** Continuously monitor and review session activity to detect any unauthorized access attempts or suspicious session behavior.

References:

- [OWASP Authentication Cheat Sheet](#)

Component: CLI Application

1 Recommended countermeasures

Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High Not tested

- State: Recommended
- Description:

To enhance the security of your CLI application, it's essential to focus on two pivotal areas: configuration management and credential management. A strategic approach to these components not only fortifies your application against unauthorized access but also ensures the safeguarding of sensitive information. Below, we detail actionable steps developers can take to implement these practices effectively:

 - **Robust Configuration Management Process:**
 - Begin by establishing a standardized process for configuring systems and applications. This includes predefined security settings that align with industry best practices and compliance requirements.
 - Regularly review and update configurations to address newly identified vulnerabilities and security threats. Utilize automated tools where possible to ensure configurations remain consistent across environments.
 - Document all configuration changes and maintain version control to enable audit trails and rollback capabilities in case of issues.
 - **Regular Security Reviews:**
 - Conduct periodic security reviews of system and application configurations to identify potential security gaps or misconfigurations.
 - Engage in penetration testing and vulnerability assessments to evaluate the effectiveness of current security configurations and identify areas for improvement.
 - **Secure Credential Management:**
 - Store credentials (e.g., passwords, API keys, secret tokens) using encrypted storage solutions. Ensure encryption keys are managed securely and access to them is tightly controlled.
 - Avoid hard-coding credentials in source code or configuration files. Instead, use secure vaults or environment variables for managing sensitive information.
 - Implement multi-factor authentication and strong password policies to enhance the security of user accounts and protect against unauthorized access.
 - Regularly rotate credentials and keys, especially after personnel changes or when a breach is suspected.

Component: Source Code Repository

1 Recommended countermeasures

Rec1. Protect access credentials in source code repositories C-SOURCE-CODE-REPOSITORY-CNT-02 Very high Not tested

- State: Recommended
- Description:

User access to repositories is often authenticated using credentials, such as passwords or private keys. Loss of these credentials may allow an attacker to gain unauthorized access.

Remediation:
Ensure that developers are encouraged to protect these credentials while they are used and managed within your development environment. Private keys should be password protected. Backing them onto a hardware token that makes them harder to abuse. Consider the option of rotating access keys.

Reference:

 - [Protect access credentials](#)

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Enforce MFA, use secure cookies, and invalidate sessions C-WEB-APPLICATION-SERVER-SIDE-CNT-02 High

Component: CLI Application

Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High

Component: Source Code Repository

Rec1. Protect access credentials in source code repositories C-SOURCE-CODE-REPOSITORY-CNT-02 Very high

PCI-DSS-v3.2.1: 8.3

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

3 Recommended countermeasures

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High Not tested

- State: Recommended
- Description:

To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.

Implementation Steps:

1. **Role-Based Access Control (RBAC):** Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.
2. **Permission Validation:** For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.
3. **Enforce Fine-Grained Access Control:** Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.
4. **Session Management:** Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.
5. **Audit Logs:** Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.

References:

- [OWASP Access Control Cheat Sheet](#)

Rec2. Enforce MFA, use secure cookies, and invalidate sessions C-WEB-APPLICATION-SERVER-SIDE-CNT-02 High Not tested

• State: Recommended

• Description:

To enhance authentication security, enforce multi-factor authentication (MFA) for all users, particularly for accessing sensitive resources. Use secure cookies to store session information and ensure they are configured with proper flags (e.g., HttpOnly, Secure) to prevent unauthorized access. Additionally, implement session invalidation after logout or inactivity to minimize the risk of session hijacking or unauthorized access.

Implementation Steps:

1. **Enforce MFA:** Configure MFA for all user accounts, requiring an additional verification step (e.g., SMS, authenticator app) beyond just username and password. This should be applied to all sensitive operations and accounts with high privileges.
2. **Use Secure Cookies:** Set cookies with the Secure flag (to ensure they are only sent over HTTPS), HttpOnly flag (to prevent access via JavaScript), and SameSite flag (to restrict cross-site request behavior), ensuring session data is protected.
3. **Invalidate Sessions on Logout or Inactivity:** Implement session expiration or timeouts for inactivity and invalidate user sessions upon logout to prevent session hijacking. Ensure that tokens are revoked immediately when no longer needed.
4. **Monitor Session Activity:** Continuously monitor and review session activity to detect any unauthorized access attempts or suspicious session behavior.

References:

- [OWASP Authentication Cheat Sheet](#)

Rec3. Use TLS for communications and protect stored data with encryption C-WEB-APPLICATION-SERVER-SIDE-CNT-05 Very high Not tested

• State: Recommended

• Description:

To secure data during transmission and at rest, ensure that all communications are protected using Transport Layer Security (TLS), and that stored data is encrypted using strong encryption algorithms. Additionally, implement robust key management practices to safeguard encryption keys and ensure that only authorized users and systems can access sensitive data.

Implementation Steps:

1. **Enforce TLS for All Communications:** Configure all communication channels, including APIs, web traffic, and data transfers, to use TLS (preferably TLS 1.2 or higher) to protect data in transit.
2. **Encrypt Stored Data:** Use strong encryption algorithms (e.g., AES-256) to encrypt sensitive data at rest, ensuring that unauthorized users cannot access or manipulate it.
3. **Implement Key Management Practices:** Use a centralized key management system (KMS) to securely generate, store, and rotate encryption keys. Enforce access controls to ensure that only authorized systems can access encryption keys.
4. **Regularly Update and Rotate Keys:** Set up automated key rotation policies to periodically update encryption keys and minimize the risk of key compromise.

References:

- [OWASP Transport Layer Security \(TLS\) Cheat Sheet](#)

Component: CLI Application

1 Recommended countermeasures

Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High Not tested

• State: Recommended

• Description:

To enhance the security of your CLI application, it's essential to focus on two pivotal areas: configuration management and credential management. A strategic approach to these components not only fortifies your application against unauthorized access but also ensures the safeguarding of sensitive information. Below, we detail actionable steps developers can take to implement these practices effectively:

- **Robust Configuration Management Process:**
 - Begin by establishing a standardized process for configuring systems and applications. This includes predefined security settings that align with industry best practices and compliance requirements.
 - Regularly review and update configurations to address newly identified vulnerabilities and security threats. Utilize automated tools where possible to ensure configurations remain consistent across environments.
 - Document all configuration changes and maintain version control to enable audit trails and rollback capabilities in case of issues.
- **Regular Security Reviews:**
 - Conduct periodic security reviews of system and application configurations to identify potential security gaps or misconfigurations.
 - Engage in penetration testing and vulnerability assessments to evaluate the effectiveness of current security configurations and identify areas for improvement.
- **Secure Credential Management:**
 - Store credentials (e.g., passwords, API keys, secret tokens) using encrypted storage solutions. Ensure encryption keys are managed securely and access to them is tightly controlled.
 - Avoid hard-coding credentials in source code or configuration files. Instead, use secure vaults or environment variables for managing sensitive information.
 - Implement multi-factor authentication and strong password policies to enhance the security of user accounts and protect against unauthorized access.
 - Regularly rotate credentials and keys, especially after personnel changes or when a breach is suspected.

Component: Source Code Repository

1 Recommended countermeasures

Rec1. Protect access credentials in source code repositories C-SOURCE-CODE-REPOSITORY-CNT-02 Very high Not tested

• State: Recommended

• Description:

User access to repositories is often authenticated using credentials, such as passwords or private keys. Loss of these credentials may allow an attacker to gain unauthorized access. Remediation:

Ensure that developers are encouraged to protect these credentials while they are used and managed within your development environment. Private keys should be password protected. Backing them onto a hardware token that makes them harder to abuse. Consider the option of rotating access keys.

Reference:

- [Protect access credentials](#)

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

- Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High
- Rec2. Enforce MFA, use secure cookies, and invalidate sessions C-WEB-APPLICATION-SERVER-SIDE-CNT-02 High
- Rec3. Use TLS for communications and protect stored data with encryption C-WEB-APPLICATION-SERVER-SIDE-CNT-05 Very high

Component: CLI Application

- Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High

Component: Source Code Repository

- Rec1. Protect access credentials in source code repositories C-SOURCE-CODE-REPOSITORY-CNT-02 Very high

PCI-DSS-v3.2.1: 8.7

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

1 Recommended countermeasures

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High Not tested

State: Recommended

Description:

To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.

Implementation Steps:

- Role-Based Access Control (RBAC):** Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.
- Permission Validation:** For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.
- Enforce Fine-Grained Access Control:** Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.
- Session Management:** Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.
- Audit Logs:** Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.

References:

- [OWASP Access Control Cheat Sheet](#)

Component: CLI Application

1 Recommended countermeasures

Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High Not tested

State: Recommended

Description:

To enhance the security of your CLI application, it's essential to focus on two pivotal areas: configuration management and credential management. A strategic approach to these components not only fortifies your application against unauthorized access but also ensures the safeguarding of sensitive information. Below, we detail actionable steps developers can take to implement these practices effectively:

- Robust Configuration Management Process:**
 - Begin by establishing a standardized process for configuring systems and applications. This includes predefined security settings that align with industry best practices and compliance requirements.
 - Regularly review and update configurations to address newly identified vulnerabilities and security threats. Utilize automated tools where possible to ensure configurations remain consistent across environments.
 - Document all configuration changes and maintain version control to enable audit trails and rollback capabilities in case of issues.
- Regular Security Reviews:**
 - Conduct periodic security reviews of system and application configurations to identify potential security gaps or misconfigurations.
 - Engage in penetration testing and vulnerability assessments to evaluate the effectiveness of current security configurations and identify areas for improvement.
- Secure Credential Management:**
 - Store credentials (e.g., passwords, API keys, secret tokens) using encrypted storage solutions. Ensure encryption keys are managed securely and access to them is tightly controlled.
 - Avoid hard-coding credentials in source code or configuration files. Instead, use secure vaults or environment variables for managing sensitive information.
 - Implement multi-factor authentication and strong password policies to enhance the security of user accounts and protect against unauthorized access.
 - Regularly rotate credentials and keys, especially after personnel changes or when a breach is suspected.

Component: Source Code Repository

1 Recommended countermeasures

Rec1. Enforce a model of least privilege for source code repositories C-SOURCE-CODE-REPOSITORY-CNT-03 High Not tested

State: Recommended

Description:

Least privilege is the principle that a security architecture should be designed so that each entity is granted the minimum system resources and authorizations that the entity needs to perform its function.

Remediation:

Consider the exposure of your repository. Technically enforce a model of least privilege for who can make changes to your code repository. Make sure all activity is attributable. Additionally, access to the repository should be revoked swiftly when no longer required, or in the event of compromise

Reference:

- [Protect access credentials](#)

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High

Component: CLI Application

Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High

Component: Source Code Repository

Rec1. Enforce a model of least privilege for source code repositories C-SOURCE-CODE-REPOSITORY-CNT-03 High

PCI-DSS-v3.2.1: 9.3

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

2 Recommended countermeasures

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High Not tested

- State: Recommended
- Description:

To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.
- Implementation Steps:
 - Role-Based Access Control (RBAC):** Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.
 - Permission Validation:** For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.
 - Enforce Fine-Grained Access Control:** Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.
 - Session Management:** Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.
 - Audit Logs:** Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.
- References:
 - [OWASP Access Control Cheat Sheet](#)

Rec2. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High Not tested

- State: Recommended
- Description:

To minimize the attack surface and enhance security, disable any unused services, remove default accounts, and apply security updates promptly. By doing so, you reduce the risk of exploitation through unneeded services or default credentials and ensure that the system is protected from known vulnerabilities. Additionally, use secure configurations to prevent unauthorized access and ensure that the system is optimally protected.
- Implementation Steps:
 - Disable Unused Services:** Identify and disable any services that are not needed for the application or system to function. This limits the number of potential attack vectors.
 - Remove Default Accounts:** Remove or disable default accounts and ensure that all active accounts are assigned strong, unique credentials. Avoid using default settings that might be easily guessed by attackers.
 - Apply Security Updates:** Regularly check for and apply security patches to all systems and applications. Set up an automated patch management process to ensure that critical updates are applied as soon as they are released.
 - Use Secure Configurations:** Ensure that all configurations, such as database, server, and application settings, follow security best practices, including strong encryption, least privilege access, and secure communication protocols.
- References:
 - [OWASP Secure Configuration Guide](#)

Component: CLI Application

1 Recommended countermeasures

Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High Not tested

- State: Recommended
- Description:

To enhance the security of your CLI application, it's essential to focus on two pivotal areas: configuration management and credential management. A strategic approach to these components not only fortifies your application against unauthorized access but also ensures the safeguarding of sensitive information. Below, we detail actionable steps developers can take to implement these practices effectively:
- Robust Configuration Management Process:
 - Begin by establishing a standardized process for configuring systems and applications. This includes predefined security settings that align with industry best practices and compliance requirements.
 - Regularly review and update configurations to address newly identified vulnerabilities and security threats. Utilize automated tools where possible to ensure configurations remain consistent across environments.
 - Document all configuration changes and maintain version control to enable audit trails and rollback capabilities in case of issues.
- Regular Security Reviews:
 - Conduct periodic security reviews of system and application configurations to identify potential security gaps or misconfigurations.
 - Engage in penetration testing and vulnerability assessments to evaluate the effectiveness of current security configurations and identify areas for improvement.
- Secure Credential Management:
 - Store credentials (e.g., passwords, API keys, secret tokens) using encrypted storage solutions. Ensure encryption keys are managed securely and access to them is tightly controlled.
 - Avoid hard-coding credentials in source code or configuration files. Instead, use secure vaults or environment variables for managing sensitive information.
 - Implement multi-factor authentication and strong password policies to enhance the security of user accounts and protect against unauthorized access.
 - Regularly rotate credentials and keys, especially after personnel changes or when a breach is suspected.

Component: Document-Oriented Database

1 Recommended countermeasures

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CM5 Very high Not tested

- State: Recommended
- Description:

Conducting regular configuration audits of a Document-Oriented Database is vital for ensuring that security settings and configurations are aligned with best practices and organizational security policies. Here's how to systematically approach these audits:

 - Establish a Baseline Configuration
 - Identify Security Settings:** Document the optimal security configurations for your database environment. This includes access controls, encryption settings, network configurations, and any specific database security features.
 - Create a Configuration Template:** Develop a template or checklist that outlines all security-related configuration settings. This template will serve as the baseline for regular audits.
 - Automate the Audit Process
 - Use Configuration Management Tools:** Implement tools that can automate the process of detecting deviations from the baseline configuration. Tools like Puppet, Chef, Ansible, or database-specific management solutions can help.
 - Schedule Regular Scans:** Configure these tools to perform regular scans of your document-oriented database environment to identify and report any configuration changes or non-compliance with the baseline.
 - Review Access Controls
 - Validate User Access:** Regularly review user accounts, roles, and permissions to ensure that they adhere to the principle of least privilege. Remove or adjust any excessive permissions.
 - Audit Authentication Mechanisms:** Verify that authentication methods (password policies, multi-factor authentication, etc.) are properly configured and enforced.
 - Verify Network Configurations
 - Check Network Access Controls:** Ensure that the database is not exposed to unauthorized networks. Validate firewall rules and other network security configurations that control access to the database.
 - Secure Communication Channels:** Confirm that data in transit is encrypted and that secure communication protocols are in use.
 - Confirm Data Protection Measures
 - Data Encryption:** Verify that data at rest is encrypted according to best practices. Check the configuration of encryption mechanisms and the management of encryption keys.
 - Backup and Recovery:** Review the configuration of backup processes to ensure that data is securely backed up and that encryption is maintained for backups.
 - Document and Resolve Findings
 - Report Findings:** Document any deviations from the baseline configuration identified during the audit. Classify findings based on their potential impact on security.
 - Plan Remediation:** Develop a plan for addressing high-priority findings. This may include reconfiguring settings, updating software, or modifying access controls.
 - Track Progress:** Maintain records of audits, findings, and remediation efforts to track improvements over time and to provide accountability.
 - Update Baseline Configurations
 - Incorporate Changes:** As you make changes to your database environment or as new security best practices emerge, update your baseline configuration and templates accordingly.
 - Review Periodically:** Regularly review the baseline configuration to ensure it remains relevant and effective in mitigating security risks.
 - Educate and Train
 - Team Awareness:** Ensure that your team is aware of the configuration audit process, the importance of adhering to the baseline configurations, and the role they play in maintaining database security.
 - Continuous Learning:** Stay informed about new security features offered by your document-oriented database system and incorporate relevant features into your security baseline.

References and Good Security Practices

 - Industry Best Practices:** Follow guidelines and security checklists provided by the database vendor, as well as security standards relevant to your industry (e.g., CIS Benchmarks, NIST Guidelines).
 - Security Communities:** Engage with security communities and forums related to your database technology to stay updated on the latest security advisories and recommendations.

Regular configuration audits are a proactive step in maintaining the security and integrity of your document-oriented database, helping to prevent unauthorized access, data leaks, and ensuring compliance with data protection regulations.

Component: Source Code Repository

1 Recommended countermeasures

Rec1. Enforce a model of least privilege for source code repositories C-SOURCE-CODE-REPOSITORY-CNT-03 High Not tested

- State: Recommended
- Description:

Least privilege is the principle that a security architecture should be designed so that each entity is granted the minimum system resources and authorizations that the entity needs to perform its function.

Remediation:
Consider the exposure of your repository. Technically enforce a model of least privilege for who can make changes to your code repository. Make sure all activity is attributable. Additionally, access to the repository should be revoked swiftly when no longer required, or in the event of compromise

Reference:

 - [Protect access credentials](#)

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

- Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High
- Rec2. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High

Component: CLI Application

- Rec1. Comprehensive security configuration and credential management C-CLI-APPLICATION-C003 High

Component: Document-Oriented Database

- Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CM5 Very high

Component: Source Code Repository

- Rec1. Enforce a model of least privilege for source code repositories C-SOURCE-CODE-REPOSITORY-CNT-03 High

PCI-DSS-v3.2.1: 6.1

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: CLI Application

1 Recommended countermeasures

Rec1. Establish regular dependency scanning processes C-CLI-APPLICATION-C001 Very high Not tested

- State: Recommended
- Description:

In the realm of Command-Line Interface (CLI) application development, managing dependencies is a critical aspect of maintaining security and functionality. Dependencies, such as libraries and frameworks, can introduce vulnerabilities into your application if they are outdated or compromised. To mitigate this risk, implementing a process for regular dependency scanning is essential. This process involves:

 - Automated Scanning:** Utilizing automated tools, such as OWASP Dependency-Check, to systematically scan your CLI application's dependencies. These tools can identify known vulnerabilities by comparing your project's dependencies against public vulnerability databases.
 - Continuous Integration:** Integrating dependency scanning into your continuous integration (CI) pipeline ensures that scans are performed regularly, ideally with every build or update. This practice helps in early detection of vulnerabilities as they arise.
 - Timely Updates:** Upon identification of vulnerable dependencies, it's crucial to update them to the latest, secure versions promptly. If no updates are available, consider replacing the dependency with a more secure alternative.
 - Version Management:** Maintaining an inventory of dependencies and their versions helps in tracking vulnerabilities and ensuring that all components are up-to-date.
 - Vulnerability Alerting:** Configuring alerting mechanisms within scanning tools or CI systems to notify developers of detected vulnerabilities. This allows for immediate action to remediate or mitigate risks.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: CLI Application

Rec1. Establish regular dependency scanning processes C-CLI-APPLICATION-C001 Very high

PCI-DSS-v3.2.1: 6.2

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

2 Recommended countermeasures

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High Not tested

- State: Recommended
- Description:

To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.

Implementation Steps:

 - Role-Based Access Control (RBAC):** Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.
 - Permission Validation:** For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.
 - Enforce Fine-Grained Access Control:** Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.
 - Session Management:** Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.
 - Audit Logs:** Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.

References:

 - [OWASP Access Control Cheat Sheet](#)

Rec2. Block or validate all outbound requests C-WEB-APPLICATION-SERVER-SIDE-CNT-11 Medium Not tested

- State: Recommended
- Description:

To reduce the risk of data exfiltration or unauthorized system communication, block or carefully validate all outbound requests made by the application. Ensure that only requests to specific, trusted domains and IP addresses are allowed, and that any outbound request to unapproved destinations is blocked or flagged for further review. This control prevents the application from communicating with malicious external systems and protects sensitive data from being leaked.

Implementation Steps:

 - Implement Outbound Request Filtering:** Use a web application firewall (WAF) or proxy to block or monitor any outbound requests that are not to trusted domains or IP addresses.
 - Validate Outbound Requests:** Before allowing outbound communication, validate the destination domain and IP address to ensure they are within an approved list of resources necessary for the application.
 - Set Domain and IP Allowlists:** Define an allowlist of specific domains and IP addresses that the application is authorized to interact with, and block all other outbound traffic.
 - Monitor and Log Outbound Requests:** Continuously monitor and log all outbound requests for suspicious activity or attempts to communicate with unapproved destinations.
 - Review and Update Allowlist Regularly:** Periodically review the allowlist to ensure that it remains up-to-date with the application's legitimate requirements.

References:

 - [Security and Privacy Controls for Information Systems and Organizations](#)

Component: CLI Application

1 Recommended countermeasures

Rec1. Establish regular dependency scanning processes C-CLI-APPLICATION-C001 Very high Not tested

- State: Recommended
- Description:

In the realm of Command-Line Interface (CLI) application development, managing dependencies is a critical aspect of maintaining security and functionality. Dependencies, such as libraries and frameworks, can introduce vulnerabilities into your application if they are outdated or compromised. To mitigate this risk, implementing a process for regular dependency scanning is essential. This process involves:

 - Automated Scanning:** Utilizing automated tools, such as OWASP Dependency-Check, to systematically scan your CLI application's dependencies. These tools can identify known vulnerabilities by comparing your project's dependencies against public vulnerability databases.
 - Continuous Integration:** Integrating dependency scanning into your continuous integration (CI) pipeline ensures that scans are performed regularly, ideally with every build or update. This practice helps in early detection of vulnerabilities as they arise.
 - Timely Updates:** Upon identification of vulnerable dependencies, it's crucial to update them to the latest, secure versions promptly. If no updates are available, consider replacing the dependency with a more secure alternative.

- **Version Management:** Maintaining an inventory of dependencies and their versions helps in tracking vulnerabilities and ensuring that all components are up-to-date.
- **Vulnerability Alerting:** Configuring alerting mechanisms within scanning tools or CI systems to notify developers of detected vulnerabilities. This allows for immediate action to remediate or mitigate risks.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

- Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High
- Rec2. Block or validate all outbound requests C-WEB-APPLICATION-SERVER-SIDE-CNT-11 Medium

Component: CLI Application

- Rec1. Establish regular dependency scanning processes C-CLI-APPLICATION-C001 Very high

PCI-DSS-v3.2.1: 6.5

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: CLI Application

1 Recommended countermeasures

- Rec1. Establish regular dependency scanning processes C-CLI-APPLICATION-C001 Very high Not tested

- **State:** Recommended
- **Description:**

In the realm of Command-Line Interface (CLI) application development, managing dependencies is a critical aspect of maintaining security and functionality. Dependencies, such as libraries and frameworks, can introduce vulnerabilities into your application if they are outdated or compromised. To mitigate this risk, implementing a process for regular dependency scanning is essential. This process involves:

 - **Automated Scanning:** Utilizing automated tools, such as OWASP Dependency-Check, to systematically scan your CLI application's dependencies. These tools can identify known vulnerabilities by comparing your project's dependencies against public vulnerability databases.
 - **Continuous Integration:** Integrating dependency scanning into your continuous integration (CI) pipeline ensures that scans are performed regularly, ideally with every build or update. This practice helps in early detection of vulnerabilities as they arise.
 - **Timely Updates:** Upon identification of vulnerable dependencies, it's crucial to update them to the latest, secure versions promptly. If no updates are available, consider replacing the dependency with a more secure alternative.
 - **Version Management:** Maintaining an inventory of dependencies and their versions helps in tracking vulnerabilities and ensuring that all components are up-to-date.
 - **Vulnerability Alerting:** Configuring alerting mechanisms within scanning tools or CI systems to notify developers of detected vulnerabilities. This allows for immediate action to remediate or mitigate risks.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: CLI Application

- Rec1. Establish regular dependency scanning processes C-CLI-APPLICATION-C001 Very high

PCI-DSS-v3.2.1: 11.2

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: CLI Application

1 Recommended countermeasures

- Rec1. Establish regular dependency scanning processes C-CLI-APPLICATION-C001 Very high Not tested

- **State:** Recommended
- **Description:**

In the realm of Command-Line Interface (CLI) application development, managing dependencies is a critical aspect of maintaining security and functionality. Dependencies, such as libraries and frameworks, can introduce vulnerabilities into your application if they are outdated or compromised. To mitigate this risk, implementing a process for regular dependency scanning is essential. This process involves:

 - **Automated Scanning:** Utilizing automated tools, such as OWASP Dependency-Check, to systematically scan your CLI application's dependencies. These tools can identify known vulnerabilities by comparing your project's dependencies against public vulnerability databases.
 - **Continuous Integration:** Integrating dependency scanning into your continuous integration (CI) pipeline ensures that scans are performed regularly, ideally with every build or update. This practice helps in early detection of vulnerabilities as they arise.
 - **Timely Updates:** Upon identification of vulnerable dependencies, it's crucial to update them to the latest, secure versions promptly. If no updates are available, consider replacing the dependency with a more secure alternative.
 - **Version Management:** Maintaining an inventory of dependencies and their versions helps in tracking vulnerabilities and ensuring that all components are up-to-date.
 - **Vulnerability Alerting:** Configuring alerting mechanisms within scanning tools or CI systems to notify developers of detected vulnerabilities. This allows for immediate action to remediate or mitigate risks.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: CLI Application

Rec1. Establish regular dependency scanning processes C-CLI-APPLICATION-C001 Very high

PCI-DSS-v3.2.1: 11.3

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side	
2 Recommended countermeasures	
<div>Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High Not tested</div> <div><div><div>State: Recommended</div><div>Description:<div>To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.</div></div></div><div>Implementation Steps:<div><div>1. Role-Based Access Control (RBAC): Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.</div><div>2. Permission Validation: For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.</div><div>3. Enforce Fine-Grained Access Control: Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.</div><div>4. Session Management: Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.</div><div>5. Audit Logs: Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.</div></div></div><div>References:<div><div>OWASP Access Control Cheat Sheet</div></div></div></div>	
<div>Rec2. Block or validate all outbound requests C-WEB-APPLICATION-SERVER-SIDE-CNT-11 Medium Not tested</div> <div><div><div>State: Recommended</div><div>Description:<div>To reduce the risk of data exfiltration or unauthorized system communication, block or carefully validate all outbound requests made by the application. Ensure that only requests to specific, trusted domains and IP addresses are allowed, and that any outbound request to unapproved destinations is blocked or flagged for further review. This control prevents the application from communicating with malicious external systems and protects sensitive data from being leaked.</div></div></div><div>Implementation Steps:<div><div>1. Implement Outbound Request Filtering: Use a web application firewall (WAF) or proxy to block or monitor any outbound requests that are not to trusted domains or IP addresses.</div><div>2. Validate Outbound Requests: Before allowing outbound communication, validate the destination domain and IP address to ensure they are within an approved list of resources necessary for the application.</div><div>3. Set Domain and IP Allowlists: Define an allowlist of specific domains and IP addresses that the application is authorized to interact with, and block all other outbound traffic.</div><div>4. Monitor and Log Outbound Requests: Continuously monitor and log all outbound requests for suspicious activity or attempts to communicate with unapproved destinations.</div><div>5. Review and Update Allowlist Regularly: Periodically review the allowlist to ensure that it remains up-to-date with the application's legitimate requirements.</div></div></div><div>References:<div><div>Security and Privacy Controls for Information Systems and Organizations</div></div></div></div>	
Component: CLI Application	
1 Recommended countermeasures	
<div>Rec1. Establish regular dependency scanning processes C-CLI-APPLICATION-C001 Very high Not tested</div> <div><div><div>State: Recommended</div><div>Description:<div>In the realm of Command-Line Interface (CLI) application development, managing dependencies is a critical aspect of maintaining security and functionality. Dependencies, such as libraries and frameworks, can introduce vulnerabilities into your application if they are outdated or compromised. To mitigate this risk, implementing a process for regular dependency scanning is essential. This process involves:<div><div>Automated Scanning: Utilizing automated tools, such as OWASP Dependency-Check, to systematically scan your CLI application's dependencies. These tools can identify known vulnerabilities by comparing your project's dependencies against public vulnerability databases.</div><div>Continuous Integration: Integrating dependency scanning into your continuous integration (CI) pipeline ensures that scans are performed regularly, ideally with every build or update. This practice helps in early detection of vulnerabilities as they arise.</div><div>Timely Updates: Upon identification of vulnerable dependencies, it's crucial to update them to the latest, secure versions promptly. If no updates are available, consider replacing the dependency with a more secure alternative.</div><div>Version Management: Maintaining an inventory of dependencies and their versions helps in tracking vulnerabilities and ensuring that all components are up-to-date.</div><div>Vulnerability Alerting: Configuring alerting mechanisms within scanning tools or CI systems to notify developers of detected vulnerabilities. This allows for immediate action to remediate or mitigate risks.</div></div></div></div></div></div>	

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High

Rec2. Block or validate all outbound requests C-WEB-APPLICATION-SERVER-SIDE-CNT-11 Medium

Component: CLI Application

Rec1. Establish regular dependency scanning processes C-CLI-APPLICATION-C001 Very high

PCI-DSS-v3.2.1: 12.2

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

1 Recommended countermeasures

Rec1. Check user roles and permissions on every request

C-WEB-APPLICATION-SERVER-SIDE-CNT-03

High

Not tested

State: Recommended

Description:

To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.

Implementation Steps:

- Role-Based Access Control (RBAC):** Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.
- Permission Validation:** For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.
- Enforce Fine-Grained Access Control:** Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.
- Session Management:** Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.
- Audit Logs:** Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.

References:

- [OWASP Access Control Cheat Sheet](#)

Component: CLI Application

1 Recommended countermeasures

Rec1. Establish regular dependency scanning processes

C-CLI-APPLICATION-C001

Very high

Not tested

State: Recommended

Description:

In the realm of Command-Line Interface (CLI) application development, managing dependencies is a critical aspect of maintaining security and functionality. Dependencies, such as libraries and frameworks, can introduce vulnerabilities into your application if they are outdated or compromised. To mitigate this risk, implementing a process for regular dependency scanning is essential. This process involves:

- Automated Scanning:** Utilizing automated tools, such as OWASP Dependency-Check, to systematically scan your CLI application's dependencies. These tools can identify known vulnerabilities by comparing your project's dependencies against public vulnerability databases.
- Continuous Integration:** Integrating dependency scanning into your continuous integration (CI) pipeline ensures that scans are performed regularly, ideally with every build or update. This practice helps in early detection of vulnerabilities as they arise.
- Timely Updates:** Upon identification of vulnerable dependencies, it's crucial to update them to the latest, secure versions promptly. If no updates are available, consider replacing the dependency with a more secure alternative.
- Version Management:** Maintaining an inventory of dependencies and their versions helps in tracking vulnerabilities and ensuring that all components are up-to-date.
- Vulnerability Alerting:** Configuring alerting mechanisms within scanning tools or CI systems to notify developers of detected vulnerabilities. This allows for immediate action to remediate or mitigate risks.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Check user roles and permissions on every request

C-WEB-APPLICATION-SERVER-SIDE-CNT-03

High

Component: CLI Application

Rec1. Establish regular dependency scanning processes

C-CLI-APPLICATION-C001

Very high

PCI-DSS-v3.2.1: 8.2.1

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

1 Recommended countermeasures

Rec1. Use TLS for communications and protect stored data with encryption

C-WEB-APPLICATION-SERVER-SIDE-CNT-05

Very high

Not tested

State: Recommended

Description:

To secure data during transmission and at rest, ensure that all communications are protected using Transport Layer Security (TLS), and that stored data is encrypted using strong encryption algorithms. Additionally, implement robust key management practices to safeguard encryption keys and ensure that only authorized users and systems can access sensitive data.

Implementation Steps:

- Enforce TLS for All Communications:** Configure all communication channels, including APIs, web traffic, and data transfers, to use TLS (preferably TLS 1.2 or higher) to protect data in transit.
- Encrypt Stored Data:** Use strong encryption algorithms (e.g., AES-256) to encrypt sensitive data at rest, ensuring that unauthorized users cannot access or manipulate it.
- Implement Key Management Practices:** Use a centralized key management system (KMS) to securely generate, store, and rotate encryption keys. Enforce access controls to ensure that only authorized systems can access encryption keys.
- Regularly Update and Rotate Keys:** Set up automated key rotation policies to periodically update encryption keys and minimize the risk of key compromise.

References:

Report: CodeSentinel Thread Modeling

Compliance report - 2025-03-14T18:54:57.395420405Z

71 of 97

- OWASP Transport Layer Security (TLS) Cheat Sheet

Component: Document-Oriented Database

2 Recommended countermeasures

Rec1. Data Masking C-DOCUMENT-ORIENTED-DATABASE-CM4 Very high Not tested

- State: Recommended
- Description:

Data masking is a critical security measure for protecting sensitive information stored in a document-oriented database. It involves obscuring specific data elements within the database to prevent unauthorized users from viewing the actual content. Here's how to implement data masking effectively:

 - Identify Sensitive Data
 - Catalog Sensitive Information:** Conduct a thorough assessment of your database to identify sensitive data that requires masking. This may include personal identifiable information (PII), financial details, health records, or other confidential information.
 - Classify Data:** Categorize the data based on sensitivity levels to determine the appropriate masking techniques for each category.
 - Choose Masking Techniques
 - Static Masking:** Apply static data masking for non-production environments. Replace sensitive data with realistic, but not real, data suitable for development, testing, or training.
 - Dynamic Masking:** Implement dynamic data masking for production environments. It allows data to be masked on-the-fly, displaying masked data to unauthorized users while keeping the actual data intact and accessible for authorized users.
 - De-identification:** Remove or modify personal identifiers to protect personal privacy. This is crucial for compliance with data protection regulations.
 - Tokenization:** Use tokenization for highly sensitive data. Replace data elements with non-sensitive equivalents, known as tokens, which can be mapped back to the original data only with a specific key.
 - Implement Data Masking
 - Use Database Management Tools:** Leverage data masking features provided by your document-oriented database management system or third-party tools designed for data masking.
 - Automate Masking Processes:** Where possible, automate the data masking process to ensure consistency and efficiency. This is particularly important in dynamic environments or when dealing with large volumes of data.
 - Mask Data in Transit:** Ensure that data masking also applies to data in transit between the database and application layers to prevent exposure through interception.
 - Test Masked Data
 - Verify Masking Logic:** Test the masking implementation to ensure that the data is appropriately masked based on user roles and permissions. Confirm that masked data remains realistic and usable for its intended purpose, especially in non-production environments.
 - Performance Impact Analysis:** Assess the performance impact of data masking on your database operations and application response times. Adjust the masking strategy as needed to balance security with performance.
 - Monitor and Audit
 - Access Monitoring:** Continuously monitor access to sensitive data and the effectiveness of the masking controls. Look for unauthorized attempts to access or unmask the data.
 - Regular Audits:** Conduct regular audits of the data masking implementation to ensure it aligns with evolving data protection regulations and organizational policies.
 - Update Masking Rules as Needed
 - Stay Current:** Keep abreast of new data protection laws and security threats. Update your data masking rules and techniques accordingly to address new requirements or vulnerabilities.

References and Good Security Practices

 - Follow Best Practices:** Adhere to best practices for data masking laid out by organizations such as the Open Web Application Security Project (OWASP) and the National Institute of Standards and Technology (NIST).
 - Data Protection Regulations:** Ensure your data masking strategy complies with relevant data protection regulations like GDPR, CCPA, HIPAA, etc.

Implementing data masking in a document-oriented database helps minimize the risk of sensitive data exposure, ensuring that confidential information remains secure even in the event of unauthorized access.

Rec2. Encryption at Rest C-DOCUMENT-ORIENTED-DATABASE-CM3 Very high Not tested

- State: Recommended
- Description:

To enhance the security of your document-oriented database by protecting stored data against unauthorized access, implement encryption at rest. This involves encrypting the data files on the server so that they cannot be read without the encryption key, safeguarding sensitive information even if physical access to the storage is obtained. Follow these steps to apply encryption at rest effectively:

Step 1: Choose an Encryption Method

Research Encryption Algorithms: Opt for strong, industry-standard encryption algorithms like AES-256 to secure your data at rest. Ensure the encryption solution supports your chosen algorithm.

Select an Encryption Solution: Determine whether your document-oriented database offers native encryption at rest features. If not, consider third-party encryption tools or filesystem-level encryption solutions compatible with your operating system.

Step 2: Implement Encryption

Native Database Encryption:

 - If your database system supports native encryption at rest, enable this feature according to the database documentation. This might involve configuring database settings and restarting the database service.
 - Set up key management according to best practices, which may involve using a secure key management system (KMS) or hardware security module (HSM).

Filesystem-Level Encryption:

 - For databases without native encryption, use filesystem-level encryption tools such as dm-crypt for Linux or BitLocker for Windows. Configure these tools to encrypt the volume where your database files are stored.
 - Initialize the encryption tool, create an encrypted volume, and move your database files to this volume, following the tool's documentation.

Step 3: Manage Encryption Keys

Secure Key Storage: Store encryption keys securely, separate from the encrypted data. Use a dedicated key management service (KMS) when possible.

Key Rotation: Regularly rotate encryption keys to enhance security. Ensure your encryption solution or database system supports key rotation without data loss.

Access Controls for Keys: Implement strict access controls for encryption key management, allowing only authorized personnel to manage or access the encryption keys.

Step 4: Regular Testing and Auditing

Test Encryption Implementation: Regularly test the encryption setup to ensure data is properly encrypted and that keys are not exposed. This could involve auditing file access paths and attempting to access data without the encryption keys.

Audit Access Logs: Monitor and audit access logs for any unauthorized attempts to access encrypted data or encryption keys.

Step 5: Backup and Recovery

Encrypt Backups: Ensure that backups of your database are also encrypted at rest. Apply the same encryption standards to backups as you do to live data.

Test Recovery Processes: Regularly test backup recovery processes to ensure that encrypted data can be successfully restored and decrypted with the appropriate keys.

Additional Good Security Practices

 - Document Procedures:** Maintain detailed documentation of your encryption protocols, key management processes, and emergency procedures for key compromise situations.
 - Compliance and Legal Considerations:** Ensure your encryption at rest strategy complies with relevant data protection regulations and industry standards.
 - Educate Your Team:** Provide training for your team on the importance of encryption at rest and secure key management practices.

By following these steps, you can effectively implement encryption at rest for your document-oriented database, significantly enhancing the security of stored data against unauthorized access or breaches.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Use TLS for communications and protect stored data with encryption C-WEB-APPLICATION-SERVER-SIDE-CNT-05 Very high

Component: Document-Oriented Database

Rec1. Data Masking C-DOCUMENT-ORIENTED-DATABASE-CM4 Very high

Rec2. Encryption at Rest C-DOCUMENT-ORIENTED-DATABASE-CM3 Very high

PCI-DSS-v3.2.1: 11.1.1

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

1 Recommended countermeasures

Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High Not tested

- State: Recommended
- Description:

To minimize the attack surface and enhance security, disable any unused services, remove default accounts, and apply security updates promptly. By doing so, you reduce the risk of exploitation through unneeded services or default credentials and ensure that the system is protected from known vulnerabilities. Additionally, use secure configurations to prevent unauthorized access and ensure that the system is optimally protected.

Implementation Steps:

 - Disable Unused Services:** Identify and disable any services that are not needed for the application or system to function. This limits the number of potential attack vectors.
 - Remove Default Accounts:** Remove or disable default accounts and ensure that all active accounts are assigned strong, unique credentials. Avoid using default settings that might be easily guessed by attackers.
 - Apply Security Updates:** Regularly check for and apply security patches to all systems and applications. Set up an automated patch management process to ensure that critical updates are applied as soon as they are released.
 - Use Secure Configurations:** Ensure that all configurations, such as database, server, and application settings, follow security best practices, including strong encryption, least privilege access, and secure communication protocols.
- References:
 - [OWASP Secure Configuration Guide](#)

Component: Document-Oriented Database

1 Recommended countermeasures

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CM5 Very high Not tested

- State: Recommended
- Description:

Conducting regular configuration audits of a Document-Oriented Database is vital for ensuring that security settings and configurations are aligned with best practices and organizational security policies. Here's how to systematically approach these audits:

 - Establish a Baseline Configuration
 - Identify Security Settings:** Document the optimal security configurations for your database environment. This includes access controls, encryption settings, network configurations, and any specific database security features.
 - Create a Configuration Template:** Develop a template or checklist that outlines all security-related configuration settings. This template will serve as the baseline for regular audits.
 - Automate the Audit Process
 - Use Configuration Management Tools:** Implement tools that can automate the process of detecting deviations from the baseline configuration. Tools like Puppet, Chef, Ansible, or database-specific management solutions can help.
 - Schedule Regular Scans:** Configure these tools to perform regular scans of your document-oriented database environment to identify and report any configuration changes or non-compliance with the baseline.
 - Review Access Controls
 - Validate User Access:** Regularly review user accounts, roles, and permissions to ensure that they adhere to the principle of least privilege. Remove or adjust any excessive permissions.
 - Audit Authentication Mechanisms:** Verify that authentication methods (password policies, multi-factor authentication, etc.) are properly configured and enforced.
 - Verify Network Configurations
 - Check Network Access Controls:** Ensure that the database is not exposed to unauthorized networks. Validate firewall rules and other network security configurations that control access to the database.
 - Secure Communication Channels:** Confirm that data in transit is encrypted and that secure communication protocols are in use.
 - Confirm Data Protection Measures
 - Data Encryption:** Verify that data at rest is encrypted according to best practices. Check the configuration of encryption mechanisms and the management of encryption keys.
 - Backup and Recovery:** Review the configuration of backup processes to ensure that data is securely backed up and that encryption is maintained for backups.
 - Document and Resolve Findings
 - Report Findings:** Document any deviations from the baseline configuration identified during the audit. Classify findings based on their potential impact on security.
 - Plan Remediation:** Develop a plan for addressing high-priority findings. This may include reconfiguring settings, updating software, or modifying access controls.
 - Track Progress:** Maintain records of audits, findings, and remediation efforts to track improvements over time and to provide accountability.
 - Update Baseline Configurations
 - Incorporate Changes:** As you make changes to your database environment or as new security best practices emerge, update your baseline configuration and templates accordingly.
 - Review Periodically:** Regularly review the baseline configuration to ensure it remains relevant and effective in mitigating security risks.
 - Educate and Train
 - Team Awareness:** Ensure that your team is aware of the configuration audit process, the importance of adhering to the baseline configurations, and the role they play in maintaining database security.
 - Continuous Learning:** Stay informed about new security features offered by your document-oriented database system and incorporate relevant features into your security baseline.

References and Good Security Practices

 - Industry Best Practices:** Follow guidelines and security checklists provided by the database vendor, as well as security standards relevant to your industry (e.g., CIS Benchmarks, NIST Guidelines).
 - Security Communities:** Engage with security communities and forums related to your database technology to stay updated on the latest security advisories and recommendations.

Regular configuration audits are a proactive step in maintaining the security and integrity of your document-oriented database, helping to prevent unauthorized access, data leaks, and ensuring compliance with data protection regulations.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High

Component: Document-Oriented Database

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CM5 Very high

PCI-DSS-v3.2.1: 12.3.3

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

1 Recommended countermeasures

Rec1. Disable unused services, remove default accounts, and apply security updates

C-WEB-APPLICATION-SERVER-SIDE-CNT-06

High

Not tested

State: Recommended

Description:

To minimize the attack surface and enhance security, disable any unused services, remove default accounts, and apply security updates promptly. By doing so, you reduce the risk of exploitation through unneeded services or default credentials and ensure that the system is protected from known vulnerabilities. Additionally, use secure configurations to prevent unauthorized access and ensure that the system is optimally protected.

Implementation Steps:

- 1. Disable Unused Services:** Identify and disable any services that are not needed for the application or system to function. This limits the number of potential attack vectors.
- 2. Remove Default Accounts:** Remove or disable default accounts and ensure that all active accounts are assigned strong, unique credentials. Avoid using default settings that might be easily guessed by attackers.
- 3. Apply Security Updates:** Regularly check for and apply security patches to all systems and applications. Set up an automated patch management process to ensure that critical updates are applied as soon as they are released.
- 4. Use Secure Configurations:** Ensure that all configurations, such as database, server, and application settings, follow security best practices, including strong encryption, least privilege access, and secure communication protocols.

References:

- [OWASP Secure Configuration Guide](#)

Component: Document-Oriented Database

1 Recommended countermeasures

Rec1. Regular Configuration Audits

C-DOCUMENT-ORIENTED-DATABASE-CM5

Very high

Not tested

State: Recommended

Description:

Conducting regular configuration audits of a Document-Oriented Database is vital for ensuring that security settings and configurations are aligned with best practices and organizational security policies. Here's how to systematically approach these audits:

- 1. Establish a Baseline Configuration**
 - Identify Security Settings:** Document the optimal security configurations for your database environment. This includes access controls, encryption settings, network configurations, and any specific database security features.
 - Create a Configuration Template:** Develop a template or checklist that outlines all security-related configuration settings. This template will serve as the baseline for regular audits.
- 2. Automate the Audit Process**
 - Use Configuration Management Tools:** Implement tools that can automate the process of detecting deviations from the baseline configuration. Tools like Puppet, Chef, Ansible, or database-specific management solutions can help.
 - Schedule Regular Scans:** Configure these tools to perform regular scans of your document-oriented database environment to identify and report any configuration changes or non-compliance with the baseline.
- 3. Review Access Controls**
 - Validate User Access:** Regularly review user accounts, roles, and permissions to ensure that they adhere to the principle of least privilege. Remove or adjust any excessive permissions.
 - Audit Authentication Mechanisms:** Verify that authentication methods (password policies, multi-factor authentication, etc.) are properly configured and enforced.
- 4. Verify Network Configurations**
 - Check Network Access Controls:** Ensure that the database is not exposed to unauthorized networks. Validate firewall rules and other network security configurations that control access to the database.
 - Secure Communication Channels:** Confirm that data in transit is encrypted and that secure communication protocols are in use.
- 5. Confirm Data Protection Measures**
 - Data Encryption:** Verify that data at rest is encrypted according to best practices. Check the configuration of encryption mechanisms and the management of encryption keys.
 - Backup and Recovery:** Review the configuration of backup processes to ensure that data is securely backed up and that encryption is maintained for backups.
- 6. Document and Resolve Findings**
 - Report Findings:** Document any deviations from the baseline configuration identified during the audit. Classify findings based on their potential impact on security.
 - Plan Remediation:** Develop a plan for addressing high-priority findings. This may include reconfiguring settings, updating software, or modifying access controls.
 - Track Progress:** Maintain records of audits, findings, and remediation efforts to track improvements over time and to provide accountability.
- 7. Update Baseline Configurations**
 - Incorporate Changes:** As you make changes to your database environment or as new security best practices emerge, update your baseline configuration and templates accordingly.
 - Review Periodically:** Regularly review the baseline configuration to ensure it remains relevant and effective in mitigating security risks.
- 8. Educate and Train**
 - Team Awareness:** Ensure that your team is aware of the configuration audit process, the importance of adhering to the baseline configurations, and the role they play in maintaining database security.
 - Continuous Learning:** Stay informed about new security features offered by your document-oriented database system and incorporate relevant features into your security baseline.

References and Good Security Practices

- Industry Best Practices:** Follow guidelines and security checklists provided by the database vendor, as well as security standards relevant to your industry (e.g., CIS Benchmarks, NIST Guidelines).
- Security Communities:** Engage with security communities and forums related to your database technology to stay updated on the latest security advisories and recommendations.

Regular configuration audits are a proactive step in maintaining the security and integrity of your document-oriented database, helping to prevent unauthorized access, data leaks, and ensuring compliance with data protection regulations.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High

Component: Document-Oriented Database

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CM5 Very high

PCI-DSS-v3.2.1: 12.3.7

Report: CodeSentinel Thread Modeling

Compliance report - 2025-03-14T18:54:57.395420405Z

74 of 97

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

1 Recommended countermeasures

Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High Not tested

- State: Recommended
- Description:

To minimize the attack surface and enhance security, disable any unused services, remove default accounts, and apply security updates promptly. By doing so, you reduce the risk of exploitation through unneeded services or default credentials and ensure that the system is protected from known vulnerabilities. Additionally, use secure configurations to prevent unauthorized access and ensure that the system is optimally protected.

Implementation Steps:

 - Disable Unused Services:** Identify and disable any services that are not needed for the application or system to function. This limits the number of potential attack vectors.
 - Remove Default Accounts:** Remove or disable default accounts and ensure that all active accounts are assigned strong, unique credentials. Avoid using default settings that might be easily guessed by attackers.
 - Apply Security Updates:** Regularly check for and apply security patches to all systems and applications. Set up an automated patch management process to ensure that critical updates are applied as soon as they are released.
 - Use Secure Configurations:** Ensure that all configurations, such as database, server, and application settings, follow security best practices, including strong encryption, least privilege access, and secure communication protocols.
- References:
 - [OWASP Secure Configuration Guide](#)

Component: Document-Oriented Database

1 Recommended countermeasures

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CM5 Very high Not tested

- State: Recommended
- Description:

Conducting regular configuration audits of a Document-Oriented Database is vital for ensuring that security settings and configurations are aligned with best practices and organizational security policies. Here's how to systematically approach these audits:

 - Establish a Baseline Configuration
 - Identify Security Settings:** Document the optimal security configurations for your database environment. This includes access controls, encryption settings, network configurations, and any specific database security features.
 - Create a Configuration Template:** Develop a template or checklist that outlines all security-related configuration settings. This template will serve as the baseline for regular audits.
 - Automate the Audit Process
 - Use Configuration Management Tools:** Implement tools that can automate the process of detecting deviations from the baseline configuration. Tools like Puppet, Chef, Ansible, or database-specific management solutions can help.
 - Schedule Regular Scans:** Configure these tools to perform regular scans of your document-oriented database environment to identify and report any configuration changes or non-compliance with the baseline.
 - Review Access Controls
 - Validate User Access:** Regularly review user accounts, roles, and permissions to ensure that they adhere to the principle of least privilege. Remove or adjust any excessive permissions.
 - Audit Authentication Mechanisms:** Verify that authentication methods (password policies, multi-factor authentication, etc.) are properly configured and enforced.
 - Verify Network Configurations
 - Check Network Access Controls:** Ensure that the database is not exposed to unauthorized networks. Validate firewall rules and other network security configurations that control access to the database.
 - Secure Communication Channels:** Confirm that data in transit is encrypted and that secure communication protocols are in use.
 - Confirm Data Protection Measures
 - Data Encryption:** Verify that data at rest is encrypted according to best practices. Check the configuration of encryption mechanisms and the management of encryption keys.
 - Backup and Recovery:** Review the configuration of backup processes to ensure that data is securely backed up and that encryption is maintained for backups.
 - Document and Resolve Findings
 - Report Findings:** Document any deviations from the baseline configuration identified during the audit. Classify findings based on their potential impact on security.
 - Plan Remediation:** Develop a plan for addressing high-priority findings. This may include reconfiguring settings, updating software, or modifying access controls.
 - Track Progress:** Maintain records of audits, findings, and remediation efforts to track improvements over time and to provide accountability.
 - Update Baseline Configurations
 - Incorporate Changes:** As you make changes to your database environment or as new security best practices emerge, update your baseline configuration and templates accordingly.
 - Review Periodically:** Regularly review the baseline configuration to ensure it remains relevant and effective in mitigating security risks.
 - Educate and Train
 - Team Awareness:** Ensure that your team is aware of the configuration audit process, the importance of adhering to the baseline configurations, and the role they play in maintaining database security.
 - Continuous Learning:** Stay informed about new security features offered by your document-oriented database system and incorporate relevant features into your security baseline.
- References and Good Security Practices
 - Industry Best Practices:** Follow guidelines and security checklists provided by the database vendor, as well as security standards relevant to your industry (e.g., CIS Benchmarks, NIST Guidelines).
 - Security Communities:** Engage with security communities and forums related to your database technology to stay updated on the latest security advisories and recommendations.

Regular configuration audits are a proactive step in maintaining the security and integrity of your document-oriented database, helping to prevent unauthorized access, data leaks, and ensuring compliance with data protection regulations.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High

Component: Document-Oriented Database

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CM5 Very high

PCI-DSS-v3.2.1: 1.2

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

4 Recommended countermeasures

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High Not tested

- State: Recommended
- Description:

To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.

Implementation Steps:

 - Role-Based Access Control (RBAC):** Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.
 - Permission Validation:** For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.
 - Enforce Fine-Grained Access Control:** Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.
 - Session Management:** Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.
 - Audit Logs:** Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.
- References:
 - [OWASP Access Control Cheat Sheet](#)

Rec2. Configure XML parsers to disallow external entities and DTD processing C-WEB-APPLICATION-SERVER-SIDE-CNT-10 Very high Not tested

- State: Recommended
- Description:

To prevent XML-based attacks such as XML External Entity (XXE) and Denial of Service (DoS) attacks, configure XML parsers to disallow external entities and Document Type Definition (DTD) processing by default. This ensures that any incoming XML documents cannot trigger external requests or load external data that could compromise the system, leak sensitive data, or cause resource exhaustion.

Implementation Steps:

 - Disable External Entity Resolution:** Configure XML parsers to explicitly disable the resolution of external entities by setting options or flags that prevent the parser from fetching external resources.
 - Disable DTD Processing:** Ensure that the XML parser does not process DTDs, which can be used to define and reference external entities or cause denial-of-service attacks via large or nested DTDs.
 - Use Secure Parsers:** Ensure that you are using updated and secure XML parsing libraries that offer built-in protections against XXE and other XML-based vulnerabilities.
 - Validate XML Input:** Before parsing, validate XML input to ensure it conforms to a known schema or structure, mitigating any unexpected or malicious data.
- References:
 - [OWASP XML External Entity \(XXE\) Prevention Cheat Sheet](#)

Rec3. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High Not tested

- State: Recommended
- Description:

To minimize the attack surface and enhance security, disable any unused services, remove default accounts, and apply security updates promptly. By doing so, you reduce the risk of exploitation through unneeded services or default credentials and ensure that the system is protected from known vulnerabilities. Additionally, use secure configurations to prevent unauthorized access and ensure that the system is optimally protected.

Implementation Steps:

 - Disable Unused Services:** Identify and disable any services that are not needed for the application or system to function. This limits the number of potential attack vectors.
 - Remove Default Accounts:** Remove or disable default accounts and ensure that all active accounts are assigned strong, unique credentials. Avoid using default settings that might be easily guessed by attackers.
 - Apply Security Updates:** Regularly check for and apply security patches to all systems and applications. Set up an automated patch management process to ensure that critical updates are applied as soon as they are released.
 - Use Secure Configurations:** Ensure that all configurations, such as database, server, and application settings, follow security best practices, including strong encryption, least privilege access, and secure communication protocols.
- References:
 - [OWASP Secure Configuration Guide](#)

Rec4. Block or validate all outbound requests C-WEB-APPLICATION-SERVER-SIDE-CNT-11 Medium Not tested

- State: Recommended
- Description:

To reduce the risk of data exfiltration or unauthorized system communication, block or carefully validate all outbound requests made by the application. Ensure that only requests to specific, trusted domains and IP addresses are allowed, and that any outbound request to unapproved destinations is blocked or flagged for further review. This control prevents the application from communicating with malicious external systems and protects sensitive data from being leaked.

Implementation Steps:

 - Implement Outbound Request Filtering:** Use a web application firewall (WAF) or proxy to block or monitor any outbound requests that are not to trusted domains or IP addresses.
 - Validate Outbound Requests:** Before allowing outbound communication, validate the destination domain and IP address to ensure they are within an approved list of resources necessary for the application.
 - Set Domain and IP Allowlists:** Define an allowlist of specific domains and IP addresses that the application is authorized to interact with, and block all other outbound traffic.
 - Monitor and Log Outbound Requests:** Continuously monitor and log all outbound requests for suspicious activity or attempts to communicate with unapproved destinations.
 - Review and Update Allowlist Regularly:** Periodically review the allowlist to ensure that it remains up-to-date with the application's legitimate requirements.
- References:
 - [Security and Privacy Controls for Information Systems and Organizations](#)

Component: Document-Oriented Database

1 Recommended countermeasures

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CM5 Very high Not tested

- State: Recommended
- Description:

Conducting regular configuration audits of a Document-Oriented Database is vital for ensuring that security settings and configurations are aligned with best practices and organizational security policies. Here's how to systematically approach these audits:

 - Establish a Baseline Configuration
 - Identify Security Settings:** Document the optimal security configurations for your database environment. This includes access controls, encryption settings, network configurations, and any specific database security features.

- **Create a Configuration Template:** Develop a template or checklist that outlines all security-related configuration settings. This template will serve as the baseline for regular audits.
 - 2. Automate the Audit Process
 - **Use Configuration Management Tools:** Implement tools that can automate the process of detecting deviations from the baseline configuration. Tools like Puppet, Chef, Ansible, or database-specific management solutions can help.
 - **Schedule Regular Scans:** Configure these tools to perform regular scans of your document-oriented database environment to identify and report any configuration changes or non-compliance with the baseline.
 - 3. Review Access Controls
 - **Validate User Access:** Regularly review user accounts, roles, and permissions to ensure that they adhere to the principle of least privilege. Remove or adjust any excessive permissions.
 - **Audit Authentication Mechanisms:** Verify that authentication methods (password policies, multi-factor authentication, etc.) are properly configured and enforced.
 - 4. Verify Network Configurations
 - **Check Network Access Controls:** Ensure that the database is not exposed to unauthorized networks. Validate firewall rules and other network security configurations that control access to the database.
 - **Secure Communication Channels:** Confirm that data in transit is encrypted and that secure communication protocols are in use.
 - 5. Confirm Data Protection Measures
 - **Data Encryption:** Verify that data at rest is encrypted according to best practices. Check the configuration of encryption mechanisms and the management of encryption keys.
 - **Backup and Recovery:** Review the configuration of backup processes to ensure that data is securely backed up and that encryption is maintained for backups.
 - 6. Document and Resolve Findings
 - **Report Findings:** Document any deviations from the baseline configuration identified during the audit. Classify findings based on their potential impact on security.
 - **Plan Remediation:** Develop a plan for addressing high-priority findings. This may include reconfiguring settings, updating software, or modifying access controls.
 - **Track Progress:** Maintain records of audits, findings, and remediation efforts to track improvements over time and to provide accountability.
 - 7. Update Baseline Configurations
 - **Incorporate Changes:** As you make changes to your database environment or as new security best practices emerge, update your baseline configuration and templates accordingly.
 - **Review Periodically:** Regularly review the baseline configuration to ensure it remains relevant and effective in mitigating security risks.
 - 8. Educate and Train
 - **Team Awareness:** Ensure that your team is aware of the configuration audit process, the importance of adhering to the baseline configurations, and the role they play in maintaining database security.
 - **Continuous Learning:** Stay informed about new security features offered by your document-oriented database system and incorporate relevant features into your security baseline.
- References and Good Security Practices**
- **Industry Best Practices:** Follow guidelines and security checklists provided by the database vendor, as well as security standards relevant to your industry (e.g., CIS Benchmarks, NIST Guidelines).
 - **Security Communities:** Engage with security communities and forums related to your database technology to stay updated on the latest security advisories and recommendations.
- Regular configuration audits are a proactive step in maintaining the security and integrity of your document-oriented database, helping to prevent unauthorized access, data leaks, and ensuring compliance with data protection regulations.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

- Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High
- Rec2. Configure XML parsers to disallow external entities and DTD processing C-WEB-APPLICATION-SERVER-SIDE-CNT-10 Very high
- Rec3. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High
- Rec4. Block or validate all outbound requests C-WEB-APPLICATION-SERVER-SIDE-CNT-11 Medium

Component: Document-Oriented Database

- Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CMS Very high

PCI-DSS-v3.2.1: 2.4

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

2 Recommended countermeasures

- Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High Not tested
- State: Recommended
 - Description:

To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.
 - Implementation Steps:
 1. **Role-Based Access Control (RBAC):** Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.
 2. **Permission Validation:** For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.
 3. **Enforce Fine-Grained Access Control:** Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.
 4. **Session Management:** Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.
 5. **Audit Logs:** Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.
 - References:
 - [OWASP Access Control Cheat Sheet](#)
- Rec2. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High Not tested
- State: Recommended
 - Description:

To minimize the attack surface and enhance security, disable any unused services, remove default accounts, and apply security updates promptly. By doing so, you reduce the risk of exploitation through unneeded services or default credentials and ensure that the system is protected from known vulnerabilities. Additionally, use secure configurations to prevent unauthorized access and ensure that the system is optimally protected.
 - Implementation Steps:
 1. **Disable Unused Services:** Identify and disable any services that are not needed for the application or system to function. This limits the number of potential attack vectors.
 2. **Remove Default Accounts:** Remove or disable default accounts and ensure that all active accounts are assigned strong, unique credentials. Avoid using default settings that might be easily guessed by attackers.

3. **Apply Security Updates:** Regularly check for and apply security patches to all systems and applications. Set up an automated patch management process to ensure that critical updates are applied as soon as they are released.
4. **Use Secure Configurations:** Ensure that all configurations, such as database, server, and application settings, follow security best practices, including strong encryption, least privilege access, and secure communication protocols.

- References:
- [OWASP Secure Configuration Guide](#)

Component: Document-Oriented Database

1 Recommended countermeasures

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CM5 Very high ○ Not tested

- State: ● Recommended
- Description:

Conducting regular configuration audits of a Document-Oriented Database is vital for ensuring that security settings and configurations are aligned with best practices and organizational security policies. Here's how to systematically approach these audits:

 - Establish a Baseline Configuration
 - Identify Security Settings:** Document the optimal security configurations for your database environment. This includes access controls, encryption settings, network configurations, and any specific database security features.
 - Create a Configuration Template:** Develop a template or checklist that outlines all security-related configuration settings. This template will serve as the baseline for regular audits.
 - Automate the Audit Process
 - Use Configuration Management Tools:** Implement tools that can automate the process of detecting deviations from the baseline configuration. Tools like Puppet, Chef, Ansible, or database-specific management solutions can help.
 - Schedule Regular Scans:** Configure these tools to perform regular scans of your document-oriented database environment to identify and report any configuration changes or non-compliance with the baseline.
 - Review Access Controls
 - Validate User Access:** Regularly review user accounts, roles, and permissions to ensure that they adhere to the principle of least privilege. Remove or adjust any excessive permissions.
 - Audit Authentication Mechanisms:** Verify that authentication methods (password policies, multi-factor authentication, etc.) are properly configured and enforced.
 - Verify Network Configurations
 - Check Network Access Controls:** Ensure that the database is not exposed to unauthorized networks. Validate firewall rules and other network security configurations that control access to the database.
 - Secure Communication Channels:** Confirm that data in transit is encrypted and that secure communication protocols are in use.
 - Confirm Data Protection Measures
 - Data Encryption:** Verify that data at rest is encrypted according to best practices. Check the configuration of encryption mechanisms and the management of encryption keys.
 - Backup and Recovery:** Review the configuration of backup processes to ensure that data is securely backed up and that encryption is maintained for backups.
 - Document and Resolve Findings
 - Report Findings:** Document any deviations from the baseline configuration identified during the audit. Classify findings based on their potential impact on security.
 - Plan Remediation:** Develop a plan for addressing high-priority findings. This may include reconfiguring settings, updating software, or modifying access controls.
 - Track Progress:** Maintain records of audits, findings, and remediation efforts to track improvements over time and to provide accountability.
 - Update Baseline Configurations
 - Incorporate Changes:** As you make changes to your database environment or as new security best practices emerge, update your baseline configuration and templates accordingly.
 - Review Periodically:** Regularly review the baseline configuration to ensure it remains relevant and effective in mitigating security risks.
 - Educate and Train
 - Team Awareness:** Ensure that your team is aware of the configuration audit process, the importance of adhering to the baseline configurations, and the role they play in maintaining database security.
 - Continuous Learning:** Stay informed about new security features offered by your document-oriented database system and incorporate relevant features into your security baseline.

References and Good Security Practices

 - Industry Best Practices:** Follow guidelines and security checklists provided by the database vendor, as well as security standards relevant to your industry (e.g., CIS Benchmarks, NIST Guidelines).
 - Security Communities:** Engage with security communities and forums related to your database technology to stay updated on the latest security advisories and recommendations.

Regular configuration audits are a proactive step in maintaining the security and integrity of your document-oriented database, helping to prevent unauthorized access, data leaks, and ensuring compliance with data protection regulations.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

- Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 ^ High
- Rec2. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 ^ High

Component: Document-Oriented Database

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CM5 Very high

PCI-DSS-v3.2.1: 6.4

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

1 Recommended countermeasures

Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 ^ High ○ Not tested

- State: ● Recommended
- Description:

To minimize the attack surface and enhance security, disable any unused services, remove default accounts, and apply security updates promptly. By doing so, you reduce the risk of exploitation through unneeded services or default credentials and ensure that the system is protected from known vulnerabilities. Additionally, use secure configurations to prevent unauthorized access and ensure that the system is optimally protected.

Implementation Steps:

 - Disable Unused Services:** Identify and disable any services that are not needed for the application or system to function. This limits the number of potential attack vectors.
 - Remove Default Accounts:** Remove or disable default accounts and ensure that all active accounts are assigned strong, unique credentials. Avoid using default settings that might be easily guessed by attackers.

3. **Apply Security Updates:** Regularly check for and apply security patches to all systems and applications. Set up an automated patch management process to ensure that critical updates are applied as soon as they are released.
4. **Use Secure Configurations:** Ensure that all configurations, such as database, server, and application settings, follow security best practices, including strong encryption, least privilege access, and secure communication protocols.

- References:**
- [OWASP Secure Configuration Guide](#)

Component: Document-Oriented Database

1 Recommended countermeasures

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CM5 Very high ○ Not tested

- State: ● Recommended
- Description:

Conducting regular configuration audits of a Document-Oriented Database is vital for ensuring that security settings and configurations are aligned with best practices and organizational security policies. Here's how to systematically approach these audits:

 - Establish a Baseline Configuration
 - Identify Security Settings:** Document the optimal security configurations for your database environment. This includes access controls, encryption settings, network configurations, and any specific database security features.
 - Create a Configuration Template:** Develop a template or checklist that outlines all security-related configuration settings. This template will serve as the baseline for regular audits.
 - Automate the Audit Process
 - Use Configuration Management Tools:** Implement tools that can automate the process of detecting deviations from the baseline configuration. Tools like Puppet, Chef, Ansible, or database-specific management solutions can help.
 - Schedule Regular Scans:** Configure these tools to perform regular scans of your document-oriented database environment to identify and report any configuration changes or non-compliance with the baseline.
 - Review Access Controls
 - Validate User Access:** Regularly review user accounts, roles, and permissions to ensure that they adhere to the principle of least privilege. Remove or adjust any excessive permissions.
 - Audit Authentication Mechanisms:** Verify that authentication methods (password policies, multi-factor authentication, etc.) are properly configured and enforced.
 - Verify Network Configurations
 - Check Network Access Controls:** Ensure that the database is not exposed to unauthorized networks. Validate firewall rules and other network security configurations that control access to the database.
 - Secure Communication Channels:** Confirm that data in transit is encrypted and that secure communication protocols are in use.
 - Confirm Data Protection Measures
 - Data Encryption:** Verify that data at rest is encrypted according to best practices. Check the configuration of encryption mechanisms and the management of encryption keys.
 - Backup and Recovery:** Review the configuration of backup processes to ensure that data is securely backed up and that encryption is maintained for backups.
 - Document and Resolve Findings
 - Report Findings:** Document any deviations from the baseline configuration identified during the audit. Classify findings based on their potential impact on security.
 - Plan Remediation:** Develop a plan for addressing high-priority findings. This may include reconfiguring settings, updating software, or modifying access controls.
 - Track Progress:** Maintain records of audits, findings, and remediation efforts to track improvements over time and to provide accountability.
 - Update Baseline Configurations
 - Incorporate Changes:** As you make changes to your database environment or as new security best practices emerge, update your baseline configuration and templates accordingly.
 - Review Periodically:** Regularly review the baseline configuration to ensure it remains relevant and effective in mitigating security risks.
 - Educate and Train
 - Team Awareness:** Ensure that your team is aware of the configuration audit process, the importance of adhering to the baseline configurations, and the role they play in maintaining database security.
 - Continuous Learning:** Stay informed about new security features offered by your document-oriented database system and incorporate relevant features into your security baseline.

References and Good Security Practices

 - Industry Best Practices:** Follow guidelines and security checklists provided by the database vendor, as well as security standards relevant to your industry (e.g., CIS Benchmarks, NIST Guidelines).
 - Security Communities:** Engage with security communities and forums related to your database technology to stay updated on the latest security advisories and recommendations.

Regular configuration audits are a proactive step in maintaining the security and integrity of your document-oriented database, helping to prevent unauthorized access, data leaks, and ensuring compliance with data protection regulations.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 ⬆ High

Component: Document-Oriented Database

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CM5 Very high

PCI-DSS-v3.2.1: 9.5

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

1 Recommended countermeasures

Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 ⬆ High ○ Not tested

- State: ● Recommended
- Description:

To minimize the attack surface and enhance security, disable any unused services, remove default accounts, and apply security updates promptly. By doing so, you reduce the risk of exploitation through unneeded services or default credentials and ensure that the system is protected from known vulnerabilities. Additionally, use secure configurations to prevent unauthorized access and ensure that the system is optimally protected.

Implementation Steps:

 - Disable Unused Services:** Identify and disable any services that are not needed for the application or system to function. This limits the number of potential attack vectors.
 - Remove Default Accounts:** Remove or disable default accounts and ensure that all active accounts are assigned strong, unique credentials. Avoid using default settings that might be easily guessed by attackers.
 - Apply Security Updates:** Regularly check for and apply security patches to all systems and applications. Set up an automated patch management process to ensure that critical updates are applied as soon as they are released.

4. **Use Secure Configurations:** Ensure that all configurations, such as database, server, and application settings, follow security best practices, including strong encryption, least privilege access, and secure communication protocols.

- References:
- [OWASP Secure Configuration Guide](#)

Component: Document-Oriented Database

1 Recommended countermeasures

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CM5 Very high Not tested

- State: Recommended
- Description:

Conducting regular configuration audits of a Document-Oriented Database is vital for ensuring that security settings and configurations are aligned with best practices and organizational security policies. Here's how to systematically approach these audits:

 - Establish a Baseline Configuration
 - Identify Security Settings:** Document the optimal security configurations for your database environment. This includes access controls, encryption settings, network configurations, and any specific database security features.
 - Create a Configuration Template:** Develop a template or checklist that outlines all security-related configuration settings. This template will serve as the baseline for regular audits.
 - Automate the Audit Process
 - Use Configuration Management Tools:** Implement tools that can automate the process of detecting deviations from the baseline configuration. Tools like Puppet, Chef, Ansible, or database-specific management solutions can help.
 - Schedule Regular Scans:** Configure these tools to perform regular scans of your document-oriented database environment to identify and report any configuration changes or non-compliance with the baseline.
 - Review Access Controls
 - Validate User Access:** Regularly review user accounts, roles, and permissions to ensure that they adhere to the principle of least privilege. Remove or adjust any excessive permissions.
 - Audit Authentication Mechanisms:** Verify that authentication methods (password policies, multi-factor authentication, etc.) are properly configured and enforced.
 - Verify Network Configurations
 - Check Network Access Controls:** Ensure that the database is not exposed to unauthorized networks. Validate firewall rules and other network security configurations that control access to the database.
 - Secure Communication Channels:** Confirm that data in transit is encrypted and that secure communication protocols are in use.
 - Confirm Data Protection Measures
 - Data Encryption:** Verify that data at rest is encrypted according to best practices. Check the configuration of encryption mechanisms and the management of encryption keys.
 - Backup and Recovery:** Review the configuration of backup processes to ensure that data is securely backed up and that encryption is maintained for backups.
 - Document and Resolve Findings
 - Report Findings:** Document any deviations from the baseline configuration identified during the audit. Classify findings based on their potential impact on security.
 - Plan Remediation:** Develop a plan for addressing high-priority findings. This may include reconfiguring settings, updating software, or modifying access controls.
 - Track Progress:** Maintain records of audits, findings, and remediation efforts to track improvements over time and to provide accountability.
 - Update Baseline Configurations
 - Incorporate Changes:** As you make changes to your database environment or as new security best practices emerge, update your baseline configuration and templates accordingly.
 - Review Periodically:** Regularly review the baseline configuration to ensure it remains relevant and effective in mitigating security risks.
 - Educate and Train
 - Team Awareness:** Ensure that your team is aware of the configuration audit process, the importance of adhering to the baseline configurations, and the role they play in maintaining database security.
 - Continuous Learning:** Stay informed about new security features offered by your document-oriented database system and incorporate relevant features into your security baseline.

References and Good Security Practices

 - Industry Best Practices:** Follow guidelines and security checklists provided by the database vendor, as well as security standards relevant to your industry (e.g., CIS Benchmarks, NIST Guidelines).
 - Security Communities:** Engage with security communities and forums related to your database technology to stay updated on the latest security advisories and recommendations.

Regular configuration audits are a proactive step in maintaining the security and integrity of your document-oriented database, helping to prevent unauthorized access, data leaks, and ensuring compliance with data protection regulations.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High

Component: Document-Oriented Database

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CM5 Very high

PCI-DSS-v3.2.1: 9.6

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

1 Recommended countermeasures

Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High Not tested

- State: Recommended
- Description:

To minimize the attack surface and enhance security, disable any unused services, remove default accounts, and apply security updates promptly. By doing so, you reduce the risk of exploitation through unneeded services or default credentials and ensure that the system is protected from known vulnerabilities. Additionally, use secure configurations to prevent unauthorized access and ensure that the system is optimally protected.

Implementation Steps:

 - Disable Unused Services:** Identify and disable any services that are not needed for the application or system to function. This limits the number of potential attack vectors.
 - Remove Default Accounts:** Remove or disable default accounts and ensure that all active accounts are assigned strong, unique credentials. Avoid using default settings that might be easily guessed by attackers.
 - Apply Security Updates:** Regularly check for and apply security patches to all systems and applications. Set up an automated patch management process to ensure that critical updates are applied as soon as they are released.
 - Use Secure Configurations:** Ensure that all configurations, such as database, server, and application settings, follow security best practices, including strong encryption, least privilege access, and secure communication protocols.

- References:
- OWASP Secure Configuration Guide

Component: Document-Oriented Database

1 Recommended countermeasures

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CM5 Very high Not tested

- State: Recommended
- Description:

Conducting regular configuration audits of a Document-Oriented Database is vital for ensuring that security settings and configurations are aligned with best practices and organizational security policies. Here's how to systematically approach these audits:

 - Establish a Baseline Configuration
 - Identify Security Settings:** Document the optimal security configurations for your database environment. This includes access controls, encryption settings, network configurations, and any specific database security features.
 - Create a Configuration Template:** Develop a template or checklist that outlines all security-related configuration settings. This template will serve as the baseline for regular audits.
 - Automate the Audit Process
 - Use Configuration Management Tools:** Implement tools that can automate the process of detecting deviations from the baseline configuration. Tools like Puppet, Chef, Ansible, or database-specific management solutions can help.
 - Schedule Regular Scans:** Configure these tools to perform regular scans of your document-oriented database environment to identify and report any configuration changes or non-compliance with the baseline.
 - Review Access Controls
 - Validate User Access:** Regularly review user accounts, roles, and permissions to ensure that they adhere to the principle of least privilege. Remove or adjust any excessive permissions.
 - Audit Authentication Mechanisms:** Verify that authentication methods (password policies, multi-factor authentication, etc.) are properly configured and enforced.
 - Verify Network Configurations
 - Check Network Access Controls:** Ensure that the database is not exposed to unauthorized networks. Validate firewall rules and other network security configurations that control access to the database.
 - Secure Communication Channels:** Confirm that data in transit is encrypted and that secure communication protocols are in use.
 - Confirm Data Protection Measures
 - Data Encryption:** Verify that data at rest is encrypted according to best practices. Check the configuration of encryption mechanisms and the management of encryption keys.
 - Backup and Recovery:** Review the configuration of backup processes to ensure that data is securely backed up and that encryption is maintained for backups.
 - Document and Resolve Findings
 - Report Findings:** Document any deviations from the baseline configuration identified during the audit. Classify findings based on their potential impact on security.
 - Plan Remediation:** Develop a plan for addressing high-priority findings. This may include reconfiguring settings, updating software, or modifying access controls.
 - Track Progress:** Maintain records of audits, findings, and remediation efforts to track improvements over time and to provide accountability.
 - Update Baseline Configurations
 - Incorporate Changes:** As you make changes to your database environment or as new security best practices emerge, update your baseline configuration and templates accordingly.
 - Review Periodically:** Regularly review the baseline configuration to ensure it remains relevant and effective in mitigating security risks.
 - Educate and Train
 - Team Awareness:** Ensure that your team is aware of the configuration audit process, the importance of adhering to the baseline configurations, and the role they play in maintaining database security.
 - Continuous Learning:** Stay informed about new security features offered by your document-oriented database system and incorporate relevant features into your security baseline.

References and Good Security Practices

 - Industry Best Practices:** Follow guidelines and security checklists provided by the database vendor, as well as security standards relevant to your industry (e.g., CIS Benchmarks, NIST Guidelines).
 - Security Communities:** Engage with security communities and forums related to your database technology to stay updated on the latest security advisories and recommendations.

Regular configuration audits are a proactive step in maintaining the security and integrity of your document-oriented database, helping to prevent unauthorized access, data leaks, and ensuring compliance with data protection regulations.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High

Component: Document-Oriented Database

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CM5 Very high

PCI-DSS-v3.2.1: 9.7

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

1 Recommended countermeasures

Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High Not tested

- State: Recommended
- Description:

To minimize the attack surface and enhance security, disable any unused services, remove default accounts, and apply security updates promptly. By doing so, you reduce the risk of exploitation through unneeded services or default credentials and ensure that the system is protected from known vulnerabilities. Additionally, use secure configurations to prevent unauthorized access and ensure that the system is optimally protected.

Implementation Steps:

 - Disable Unused Services:** Identify and disable any services that are not needed for the application or system to function. This limits the number of potential attack vectors.
 - Remove Default Accounts:** Remove or disable default accounts and ensure that all active accounts are assigned strong, unique credentials. Avoid using default settings that might be easily guessed by attackers.
 - Apply Security Updates:** Regularly check for and apply security patches to all systems and applications. Set up an automated patch management process to ensure that critical updates are applied as soon as they are released.
 - Use Secure Configurations:** Ensure that all configurations, such as database, server, and application settings, follow security best practices, including strong encryption, least privilege access, and secure communication protocols.

References:

- [OWASP Secure Configuration Guide](#)

Component: Document-Oriented Database

1 Recommended countermeasures

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CM5 Very high ☐ Not tested

- **State:** Recommended
- **Description:**

Conducting regular configuration audits of a Document-Oriented Database is vital for ensuring that security settings and configurations are aligned with best practices and organizational security policies. Here's how to systematically approach these audits:

 1. Establish a Baseline Configuration
 - **Identify Security Settings:** Document the optimal security configurations for your database environment. This includes access controls, encryption settings, network configurations, and any specific database security features.
 - **Create a Configuration Template:** Develop a template or checklist that outlines all security-related configuration settings. This template will serve as the baseline for regular audits.
 2. Automate the Audit Process
 - **Use Configuration Management Tools:** Implement tools that can automate the process of detecting deviations from the baseline configuration. Tools like Puppet, Chef, Ansible, or database-specific management solutions can help.
 - **Schedule Regular Scans:** Configure these tools to perform regular scans of your document-oriented database environment to identify and report any configuration changes or non-compliance with the baseline.
 3. Review Access Controls
 - **Validate User Access:** Regularly review user accounts, roles, and permissions to ensure that they adhere to the principle of least privilege. Remove or adjust any excessive permissions.
 - **Audit Authentication Mechanisms:** Verify that authentication methods (password policies, multi-factor authentication, etc.) are properly configured and enforced.
 4. Verify Network Configurations
 - **Check Network Access Controls:** Ensure that the database is not exposed to unauthorized networks. Validate firewall rules and other network security configurations that control access to the database.
 - **Secure Communication Channels:** Confirm that data in transit is encrypted and that secure communication protocols are in use.
 5. Confirm Data Protection Measures
 - **Data Encryption:** Verify that data at rest is encrypted according to best practices. Check the configuration of encryption mechanisms and the management of encryption keys.
 - **Backup and Recovery:** Review the configuration of backup processes to ensure that data is securely backed up and that encryption is maintained for backups.
 6. Document and Resolve Findings
 - **Report Findings:** Document any deviations from the baseline configuration identified during the audit. Classify findings based on their potential impact on security.
 - **Plan Remediation:** Develop a plan for addressing high-priority findings. This may include reconfiguring settings, updating software, or modifying access controls.
 - **Track Progress:** Maintain records of audits, findings, and remediation efforts to track improvements over time and to provide accountability.
 7. Update Baseline Configurations
 - **Incorporate Changes:** As you make changes to your database environment or as new security best practices emerge, update your baseline configuration and templates accordingly.
 - **Review Periodically:** Regularly review the baseline configuration to ensure it remains relevant and effective in mitigating security risks.
 8. Educate and Train
 - **Team Awareness:** Ensure that your team is aware of the configuration audit process, the importance of adhering to the baseline configurations, and the role they play in maintaining database security.
 - **Continuous Learning:** Stay informed about new security features offered by your document-oriented database system and incorporate relevant features into your security baseline.

References and Good Security Practices

 - **Industry Best Practices:** Follow guidelines and security checklists provided by the database vendor, as well as security standards relevant to your industry (e.g., CIS Benchmarks, NIST Guidelines).
 - **Security Communities:** Engage with security communities and forums related to your database technology to stay updated on the latest security advisories and recommendations.

Regular configuration audits are a proactive step in maintaining the security and integrity of your document-oriented database, helping to prevent unauthorized access, data leaks, and ensuring compliance with data protection regulations.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High

Component: Document-Oriented Database

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CM5 Very high

PCI-DSS-v3.2.1: 9.8

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

1 Recommended countermeasures

Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High ☐ Not tested

- **State:** Recommended
- **Description:**

To minimize the attack surface and enhance security, disable any unused services, remove default accounts, and apply security updates promptly. By doing so, you reduce the risk of exploitation through unneeded services or default credentials and ensure that the system is protected from known vulnerabilities. Additionally, use secure configurations to prevent unauthorized access and ensure that the system is optimally protected.

Implementation Steps:

 1. **Disable Unused Services:** Identify and disable any services that are not needed for the application or system to function. This limits the number of potential attack vectors.
 2. **Remove Default Accounts:** Remove or disable default accounts and ensure that all active accounts are assigned strong, unique credentials. Avoid using default settings that might be easily guessed by attackers.
 3. **Apply Security Updates:** Regularly check for and apply security patches to all systems and applications. Set up an automated patch management process to ensure that critical updates are applied as soon as they are released.
 4. **Use Secure Configurations:** Ensure that all configurations, such as database, server, and application settings, follow security best practices, including strong encryption, least privilege access, and secure communication protocols.
- **References:**
 - [OWASP Secure Configuration Guide](#)

Component: Document-Oriented Database

1 Recommended countermeasures

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CMS

Very high

Not tested

- State:

Recommended
- Description:

Conducting regular configuration audits of a Document-Oriented Database is vital for ensuring that security settings and configurations are aligned with best practices and organizational security policies. Here's how to systematically approach these audits:

 - Establish a Baseline Configuration
 - Identify Security Settings:** Document the optimal security configurations for your database environment. This includes access controls, encryption settings, network configurations, and any specific database security features.
 - Create a Configuration Template:** Develop a template or checklist that outlines all security-related configuration settings. This template will serve as the baseline for regular audits.
 - Automate the Audit Process
 - Use Configuration Management Tools:** Implement tools that can automate the process of detecting deviations from the baseline configuration. Tools like Puppet, Chef, Ansible, or database-specific management solutions can help.
 - Schedule Regular Scans:** Configure these tools to perform regular scans of your document-oriented database environment to identify and report any configuration changes or non-compliance with the baseline.
 - Review Access Controls
 - Validate User Access:** Regularly review user accounts, roles, and permissions to ensure that they adhere to the principle of least privilege. Remove or adjust any excessive permissions.
 - Audit Authentication Mechanisms:** Verify that authentication methods (password policies, multi-factor authentication, etc.) are properly configured and enforced.
 - Verify Network Configurations
 - Check Network Access Controls:** Ensure that the database is not exposed to unauthorized networks. Validate firewall rules and other network security configurations that control access to the database.
 - Secure Communication Channels:** Confirm that data in transit is encrypted and that secure communication protocols are in use.
 - Confirm Data Protection Measures
 - Data Encryption:** Verify that data at rest is encrypted according to best practices. Check the configuration of encryption mechanisms and the management of encryption keys.
 - Backup and Recovery:** Review the configuration of backup processes to ensure that data is securely backed up and that encryption is maintained for backups.
 - Document and Resolve Findings
 - Report Findings:** Document any deviations from the baseline configuration identified during the audit. Classify findings based on their potential impact on security.
 - Plan Remediation:** Develop a plan for addressing high-priority findings. This may include reconfiguring settings, updating software, or modifying access controls.
 - Track Progress:** Maintain records of audits, findings, and remediation efforts to track improvements over time and to provide accountability.
 - Update Baseline Configurations
 - Incorporate Changes:** As you make changes to your database environment or as new security best practices emerge, update your baseline configuration and templates accordingly.
 - Review Periodically:** Regularly review the baseline configuration to ensure it remains relevant and effective in mitigating security risks.
 - Educate and Train
 - Team Awareness:** Ensure that your team is aware of the configuration audit process, the importance of adhering to the baseline configurations, and the role they play in maintaining database security.
 - Continuous Learning:** Stay informed about new security features offered by your document-oriented database system and incorporate relevant features into your security baseline.

References and Good Security Practices

 - Industry Best Practices:** Follow guidelines and security checklists provided by the database vendor, as well as security standards relevant to your industry (e.g., CIS Benchmarks, NIST Guidelines).
 - Security Communities:** Engage with security communities and forums related to your database technology to stay updated on the latest security advisories and recommendations.

Regular configuration audits are a proactive step in maintaining the security and integrity of your document-oriented database, helping to prevent unauthorized access, data leaks, and ensuring compliance with data protection regulations.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06

High

Component: Document-Oriented Database

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CMS

Very high

PCI-DSS-v3.2.1: 9.9

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

1 Recommended countermeasures

Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06

High

Not tested

- State:

Recommended
- Description:

To minimize the attack surface and enhance security, disable any unused services, remove default accounts, and apply security updates promptly. By doing so, you reduce the risk of exploitation through unneeded services or default credentials and ensure that the system is protected from known vulnerabilities. Additionally, use secure configurations to prevent unauthorized access and ensure that the system is optimally protected.

Implementation Steps:

 - Disable Unused Services:** Identify and disable any services that are not needed for the application or system to function. This limits the number of potential attack vectors.
 - Remove Default Accounts:** Remove or disable default accounts and ensure that all active accounts are assigned strong, unique credentials. Avoid using default settings that might be easily guessed by attackers.
 - Apply Security Updates:** Regularly check for and apply security patches to all systems and applications. Set up an automated patch management process to ensure that critical updates are applied as soon as they are released.
 - Use Secure Configurations:** Ensure that all configurations, such as database, server, and application settings, follow security best practices, including strong encryption, least privilege access, and secure communication protocols.

References:

- [OWASP Secure Configuration Guide](#)

Component: Document-Oriented Database

1 Recommended countermeasures

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CMS Very high Not tested

- State: Recommended
- Description: Conducting regular configuration audits of a Document-Oriented Database is vital for ensuring that security settings and configurations are aligned with best practices and organizational security policies. Here's how to systematically approach these audits:
 - Establish a Baseline Configuration
 - Identify Security Settings:** Document the optimal security configurations for your database environment. This includes access controls, encryption settings, network configurations, and any specific database security features.
 - Create a Configuration Template:** Develop a template or checklist that outlines all security-related configuration settings. This template will serve as the baseline for regular audits.
 - Automate the Audit Process
 - Use Configuration Management Tools:** Implement tools that can automate the process of detecting deviations from the baseline configuration. Tools like Puppet, Chef, Ansible, or database-specific management solutions can help.
 - Schedule Regular Scans:** Configure these tools to perform regular scans of your document-oriented database environment to identify and report any configuration changes or non-compliance with the baseline.
 - Review Access Controls
 - Validate User Access:** Regularly review user accounts, roles, and permissions to ensure that they adhere to the principle of least privilege. Remove or adjust any excessive permissions.
 - Audit Authentication Mechanisms:** Verify that authentication methods (password policies, multi-factor authentication, etc.) are properly configured and enforced.
 - Verify Network Configurations
 - Check Network Access Controls:** Ensure that the database is not exposed to unauthorized networks. Validate firewall rules and other network security configurations that control access to the database.
 - Secure Communication Channels:** Confirm that data in transit is encrypted and that secure communication protocols are in use.
 - Confirm Data Protection Measures
 - Data Encryption:** Verify that data at rest is encrypted according to best practices. Check the configuration of encryption mechanisms and the management of encryption keys.
 - Backup and Recovery:** Review the configuration of backup processes to ensure that data is securely backed up and that encryption is maintained for backups.
 - Document and Resolve Findings
 - Report Findings:** Document any deviations from the baseline configuration identified during the audit. Classify findings based on their potential impact on security.
 - Plan Remediation:** Develop a plan for addressing high-priority findings. This may include reconfiguring settings, updating software, or modifying access controls.
 - Track Progress:** Maintain records of audits, findings, and remediation efforts to track improvements over time and to provide accountability.
 - Update Baseline Configurations
 - Incorporate Changes:** As you make changes to your database environment or as new security best practices emerge, update your baseline configuration and templates accordingly.
 - Review Periodically:** Regularly review the baseline configuration to ensure it remains relevant and effective in mitigating security risks.
 - Educate and Train
 - Team Awareness:** Ensure that your team is aware of the configuration audit process, the importance of adhering to the baseline configurations, and the role they play in maintaining database security.
 - Continuous Learning:** Stay informed about new security features offered by your document-oriented database system and incorporate relevant features into your security baseline.

References and Good Security Practices

 - Industry Best Practices:** Follow guidelines and security checklists provided by the database vendor, as well as security standards relevant to your industry (e.g., CIS Benchmarks, NIST Guidelines).
 - Security Communities:** Engage with security communities and forums related to your database technology to stay updated on the latest security advisories and recommendations.

Regular configuration audits are a proactive step in maintaining the security and integrity of your document-oriented database, helping to prevent unauthorized access, data leaks, and ensuring compliance with data protection regulations.

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Disable unused services, remove default accounts, and apply security updates C-WEB-APPLICATION-SERVER-SIDE-CNT-06 High

Component: Document-Oriented Database

Rec1. Regular Configuration Audits C-DOCUMENT-ORIENTED-DATABASE-CMS Very high

PCI-DSS-v3.2.1: 2.1

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

1 Recommended countermeasures

Rec1. Enforce MFA, use secure cookies, and invalidate sessions C-WEB-APPLICATION-SERVER-SIDE-CNT-02 High Not tested

- State: Recommended
- Description: To enhance authentication security, enforce multi-factor authentication (MFA) for all users, particularly for accessing sensitive resources. Use secure cookies to store session information and ensure they are configured with proper flags (e.g., HttpOnly, Secure) to prevent unauthorized access. Additionally, implement session invalidation after logout or inactivity to minimize the risk of session hijacking or unauthorized access.

Implementation Steps:

 - Enforce MFA:** Configure MFA for all user accounts, requiring an additional verification step (e.g., SMS, authenticator app) beyond just username and password. This should be applied to all sensitive operations and accounts with high privileges.
 - Use Secure Cookies:** Set cookies with the Secure flag (to ensure they are only sent over HTTPS), HttpOnly flag (to prevent access via JavaScript), and SameSite flag (to restrict cross-site request behavior), ensuring session data is protected.
 - Invalidate Sessions on Logout or Inactivity:** Implement session expiration or timeouts for inactivity and invalidate user sessions upon logout to prevent session hijacking. Ensure that tokens are revoked immediately when no longer needed.
 - Monitor Session Activity:** Continuously monitor and review session activity to detect any unauthorized access attempts or suspicious session behavior.

References:

 - [OWASP Authentication Cheat Sheet](#)

Component: Source Code Repository

1 Recommended countermeasures

Rec1. Protect access credentials in source code repositories C-SOURCE-CODE-REPOSITORY-CNT-02

Very high

Not tested

State: Recommended

Description:
User access to repositories is often authenticated using credentials, such as passwords or private keys. Loss of these credentials may allow an attacker to gain unauthorized access.
Remediation:
Ensure that developers are encouraged to protect these credentials while they are used and managed within your development environment.
Private keys should be password protected. Backing them onto a hardware token that makes them harder to abuse.
Consider the option of rotating access keys.
Reference:

- Protect access credentials

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Enforce MFA, use secure cookies, and invalidate sessions C-WEB-APPLICATION-SERVER-SIDE-CNT-02

High

Component: Source Code Repository

Rec1. Protect access credentials in source code repositories C-SOURCE-CODE-REPOSITORY-CNT-02

Very high

PCI-DSS-v3.2.1: 8.5

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

1 Recommended countermeasures

Rec1. Enforce MFA, use secure cookies, and invalidate sessions C-WEB-APPLICATION-SERVER-SIDE-CNT-02

High

Not tested

State: Recommended

Description:
To enhance authentication security, enforce multi-factor authentication (MFA) for all users, particularly for accessing sensitive resources. Use secure cookies to store session information and ensure they are configured with proper flags (e.g., HttpOnly, Secure) to prevent unauthorized access. Additionally, implement session invalidation after logout or inactivity to minimize the risk of session hijacking or unauthorized access.
Implementation Steps:

- Enforce MFA:** Configure MFA for all user accounts, requiring an additional verification step (e.g., SMS, authenticator app) beyond just username and password. This should be applied to all sensitive operations and accounts with high privileges.
- Use Secure Cookies:** Set cookies with the Secure flag (to ensure they are only sent over HTTPS), HttpOnly flag (to prevent access via JavaScript), and SameSite flag (to restrict cross-site request behavior), ensuring session data is protected.
- Invalidate Sessions on Logout or Inactivity:** Implement session expiration or timeouts for inactivity and invalidate user sessions upon logout to prevent session hijacking. Ensure that tokens are revoked immediately when no longer needed.
- Monitor Session Activity:** Continuously monitor and review session activity to detect any unauthorized access attempts or suspicious session behavior.

References:

- OWASP Authentication Cheat Sheet

Component: Source Code Repository

1 Recommended countermeasures

Rec1. Protect access credentials in source code repositories C-SOURCE-CODE-REPOSITORY-CNT-02

Very high

Not tested

State: Recommended

Description:
User access to repositories is often authenticated using credentials, such as passwords or private keys. Loss of these credentials may allow an attacker to gain unauthorized access.
Remediation:
Ensure that developers are encouraged to protect these credentials while they are used and managed within your development environment.
Private keys should be password protected. Backing them onto a hardware token that makes them harder to abuse.
Consider the option of rotating access keys.
Reference:

- Protect access credentials

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Enforce MFA, use secure cookies, and invalidate sessions C-WEB-APPLICATION-SERVER-SIDE-CNT-02

High

Component: Source Code Repository

Rec1. Protect access credentials in source code repositories C-SOURCE-CODE-REPOSITORY-CNT-02

Very high

PCI-DSS-v3.2.1: 8.6

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

1 Recommended countermeasures

Rec1. Enforce MFA, use secure cookies, and invalidate sessions C-WEB-APPLICATION-SERVER-SIDE-CNT-02 High Not tested

- State: Recommended
- Description:

To enhance authentication security, enforce multi-factor authentication (MFA) for all users, particularly for accessing sensitive resources. Use secure cookies to store session information and ensure they are configured with proper flags (e.g., HttpOnly, Secure) to prevent unauthorized access. Additionally, implement session invalidation after logout or inactivity to minimize the risk of session hijacking or unauthorized access.

Implementation Steps:

 - Enforce MFA:** Configure MFA for all user accounts, requiring an additional verification step (e.g., SMS, authenticator app) beyond just username and password. This should be applied to all sensitive operations and accounts with high privileges.
 - Use Secure Cookies:** Set cookies with the Secure flag (to ensure they are only sent over HTTPS), HttpOnly flag (to prevent access via JavaScript), and SameSite flag (to restrict cross-site request behavior), ensuring session data is protected.
 - Invalidate Sessions on Logout or Inactivity:** Implement session expiration or timeouts for inactivity and invalidate user sessions upon logout to prevent session hijacking. Ensure that tokens are revoked immediately when no longer needed.
 - Monitor Session Activity:** Continuously monitor and review session activity to detect any unauthorized access attempts or suspicious session behavior.

References:

 - [OWASP Authentication Cheat Sheet](#)

Component: Source Code Repository

1 Recommended countermeasures

Rec1. Protect access credentials in source code repositories C-SOURCE-CODE-REPOSITORY-CNT-02 Very high Not tested

- State: Recommended
- Description:

User access to repositories is often authenticated using credentials, such as passwords or private keys. Loss of these credentials may allow an attacker to gain unauthorized access.

Remediation:

Ensure that developers are encouraged to protect these credentials while they are used and managed within your development environment. Private keys should be password protected. Backing them onto a hardware token that makes them harder to abuse. Consider the option of rotating access keys.

Reference:

 - [Protect access credentials](#)

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Enforce MFA, use secure cookies, and invalidate sessions C-WEB-APPLICATION-SERVER-SIDE-CNT-02 High

Component: Source Code Repository

Rec1. Protect access credentials in source code repositories C-SOURCE-CODE-REPOSITORY-CNT-02 Very high

PCI-DSS-v3.2.1: 12.3

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

1 Recommended countermeasures

Rec1. Enforce MFA, use secure cookies, and invalidate sessions C-WEB-APPLICATION-SERVER-SIDE-CNT-02 High Not tested

- State: Recommended
- Description:

To enhance authentication security, enforce multi-factor authentication (MFA) for all users, particularly for accessing sensitive resources. Use secure cookies to store session information and ensure they are configured with proper flags (e.g., HttpOnly, Secure) to prevent unauthorized access. Additionally, implement session invalidation after logout or inactivity to minimize the risk of session hijacking or unauthorized access.

Implementation Steps:

 - Enforce MFA:** Configure MFA for all user accounts, requiring an additional verification step (e.g., SMS, authenticator app) beyond just username and password. This should be applied to all sensitive operations and accounts with high privileges.
 - Use Secure Cookies:** Set cookies with the Secure flag (to ensure they are only sent over HTTPS), HttpOnly flag (to prevent access via JavaScript), and SameSite flag (to restrict cross-site request behavior), ensuring session data is protected.
 - Invalidate Sessions on Logout or Inactivity:** Implement session expiration or timeouts for inactivity and invalidate user sessions upon logout to prevent session hijacking. Ensure that tokens are revoked immediately when no longer needed.
 - Monitor Session Activity:** Continuously monitor and review session activity to detect any unauthorized access attempts or suspicious session behavior.

References:

 - [OWASP Authentication Cheat Sheet](#)

Component: Source Code Repository

1 Recommended countermeasures

Rec1. Protect access credentials in source code repositories C-SOURCE-CODE-REPOSITORY-CNT-02 Very high Not tested

- State: Recommended
- Description: User access to repositories is often authenticated using credentials, such as passwords or private keys. Loss of these credentials may allow an attacker to gain unauthorized access.
Remediation: Ensure that developers are encouraged to protect these credentials while they are used and managed within your development environment. Private keys should be password protected. Backing them onto a hardware token that makes them harder to abuse. Consider the option of rotating access keys.
Reference:
 - Protect access credentials

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Enforce MFA, use secure cookies, and invalidate sessions C-WEB-APPLICATION-SERVER-SIDE-CNT-02 High

Component: Source Code Repository

Rec1. Protect access credentials in source code repositories C-SOURCE-CODE-REPOSITORY-CNT-02 Very high

PCI-DSS-v3.2.1: 1.1.1

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

3 Recommended countermeasures

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High Not tested

- State: Recommended
- Description: To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.
Implementation Steps:
 - Role-Based Access Control (RBAC): Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.
 - Permission Validation: For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.
 - Enforce Fine-Grained Access Control: Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.
 - Session Management: Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.
 - Audit Logs: Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.
References:
 - OWASP Access Control Cheat Sheet

Rec2. Configure XML parsers to disallow external entities and DTD processing C-WEB-APPLICATION-SERVER-SIDE-CNT-10 Very high Not tested

- State: Recommended
- Description: To prevent XML-based attacks such as XML External Entity (XXE) and Denial of Service (DoS) attacks, configure XML parsers to disallow external entities and Document Type Definition (DTD) processing by default. This ensures that any incoming XML documents cannot trigger external requests or load external data that could compromise the system, leak sensitive data, or cause resource exhaustion.
Implementation Steps:
 - Disable External Entity Resolution: Configure XML parsers to explicitly disable the resolution of external entities by setting options or flags that prevent the parser from fetching external resources.
 - Disable DTD Processing: Ensure that the XML parser does not process DTDs, which can be used to define and reference external entities or cause denial-of-service attacks via large or nested DTDs.
 - Use Secure Parsers: Ensure that you are using updated and secure XML parsing libraries that offer built-in protections against XXE and other XML-based vulnerabilities.
 - Validate XML Input: Before parsing, validate XML input to ensure it conforms to a known schema or structure, mitigating any unexpected or malicious data.
References:
 - OWASP XML External Entity (XXE) Prevention Cheat Sheet

Rec3. Use TLS for communications and protect stored data with encryption C-WEB-APPLICATION-SERVER-SIDE-CNT-05 Very high Not tested

- State: Recommended
- Description: To secure data during transmission and at rest, ensure that all communications are protected using Transport Layer Security (TLS), and that stored data is encrypted using strong encryption algorithms. Additionally, implement robust key management practices to safeguard encryption keys and ensure that only authorized users and systems can access sensitive data.
Implementation Steps:
 - Enforce TLS for All Communications: Configure all communication channels, including APIs, web traffic, and data transfers, to use TLS (preferably TLS 1.2 or higher) to protect data in transit.
 - Encrypt Stored Data: Use strong encryption algorithms (e.g., AES-256) to encrypt sensitive data at rest, ensuring that unauthorized users cannot access or manipulate it.
 - Implement Key Management Practices: Use a centralized key management system (KMS) to securely generate, store, and rotate encryption keys. Enforce access controls to ensure that only authorized systems can access encryption keys.
 - Regularly Update and Rotate Keys: Set up automated key rotation policies to periodically update encryption keys and minimize the risk of key compromise.
References:
 - OWASP Transport Layer Security (TLS) Cheat Sheet

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

- Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High
- Rec2. Configure XML parsers to disallow external entities and DTD processing C-WEB-APPLICATION-SERVER-SIDE-CNT-10 Very high
- Rec3. Use TLS for communications and protect stored data with encryption C-WEB-APPLICATION-SERVER-SIDE-CNT-05 Very high

PCI-DSS-v3.2.1: 1.1.2

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

3 Recommended countermeasures

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High Not tested

State: Recommended

Description:

To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.

Implementation Steps:

- Role-Based Access Control (RBAC):** Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.
- Permission Validation:** For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.
- Enforce Fine-Grained Access Control:** Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.
- Session Management:** Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.
- Audit Logs:** Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.

References:

- [OWASP Access Control Cheat Sheet](#)

Rec2. Configure XML parsers to disallow external entities and DTD processing C-WEB-APPLICATION-SERVER-SIDE-CNT-10 Very high Not tested

State: Recommended

Description:

To prevent XML-based attacks such as XML External Entity (XXE) and Denial of Service (DoS) attacks, configure XML parsers to disallow external entities and Document Type Definition (DTD) processing by default. This ensures that any incoming XML documents cannot trigger external requests or load external data that could compromise the system, leak sensitive data, or cause resource exhaustion.

Implementation Steps:

- Disable External Entity Resolution:** Configure XML parsers to explicitly disable the resolution of external entities by setting options or flags that prevent the parser from fetching external resources.
- Disable DTD Processing:** Ensure that the XML parser does not process DTDs, which can be used to define and reference external entities or cause denial-of-service attacks via large or nested DTDs.
- Use Secure Parsers:** Ensure that you are using updated and secure XML parsing libraries that offer built-in protections against XXE and other XML-based vulnerabilities.
- Validate XML Input:** Before parsing, validate XML input to ensure it conforms to a known schema or structure, mitigating any unexpected or malicious data.

References:

- [OWASP XML External Entity \(XXE\) Prevention Cheat Sheet](#)

Rec3. Use TLS for communications and protect stored data with encryption C-WEB-APPLICATION-SERVER-SIDE-CNT-05 Very high Not tested

State: Recommended

Description:

To secure data during transmission and at rest, ensure that all communications are protected using Transport Layer Security (TLS), and that stored data is encrypted using strong encryption algorithms. Additionally, implement robust key management practices to safeguard encryption keys and ensure that only authorized users and systems can access sensitive data.

Implementation Steps:

- Enforce TLS for All Communications:** Configure all communication channels, including APIs, web traffic, and data transfers, to use TLS (preferably TLS 1.2 or higher) to protect data in transit.
- Encrypt Stored Data:** Use strong encryption algorithms (e.g., AES-256) to encrypt sensitive data at rest, ensuring that unauthorized users cannot access or manipulate it.
- Implement Key Management Practices:** Use a centralized key management system (KMS) to securely generate, store, and rotate encryption keys. Enforce access controls to ensure that only authorized systems can access encryption keys.
- Regularly Update and Rotate Keys:** Set up automated key rotation policies to periodically update encryption keys and minimize the risk of key compromise.

References:

- [OWASP Transport Layer Security \(TLS\) Cheat Sheet](#)

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High

Rec2. Configure XML parsers to disallow external entities and DTD processing C-WEB-APPLICATION-SERVER-SIDE-CNT-10 Very high

Rec3. Use TLS for communications and protect stored data with encryption C-WEB-APPLICATION-SERVER-SIDE-CNT-05 Very high

PCI-DSS-v3.2.1: 1.1.3

Report: CodeSentinel Thread Modeling

Compliance report - 2025-03-14T18:54:57.395420405Z

88 of 97

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

3 Recommended countermeasures

Rec1. Check user roles and permissions on every request

C-WEB-APPLICATION-SERVER-SIDE-CNT-03

High

Not tested

State: Recommended

Description:
To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.
Implementation Steps:

- Role-Based Access Control (RBAC):** Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.
- Permission Validation:** For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.
- Enforce Fine-Grained Access Control:** Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.
- Session Management:** Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.
- Audit Logs:** Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.

References:

- [OWASP Access Control Cheat Sheet](#)

Rec2. Configure XML parsers to disallow external entities and DTD processing

C-WEB-APPLICATION-SERVER-SIDE-CNT-10

Very high

Not tested

State: Recommended

Description:
To prevent XML-based attacks such as XML External Entity (XXE) and Denial of Service (DoS) attacks, configure XML parsers to disallow external entities and Document Type Definition (DTD) processing by default. This ensures that any incoming XML documents cannot trigger external requests or load external data that could compromise the system, leak sensitive data, or cause resource exhaustion.
Implementation Steps:

- Disable External Entity Resolution:** Configure XML parsers to explicitly disable the resolution of external entities by setting options or flags that prevent the parser from fetching external resources.
- Disable DTD Processing:** Ensure that the XML parser does not process DTDs, which can be used to define and reference external entities or cause denial-of-service attacks via large or nested DTDs.
- Use Secure Parsers:** Ensure that you are using updated and secure XML parsing libraries that offer built-in protections against XXE and other XML-based vulnerabilities.
- Validate XML Input:** Before parsing, validate XML input to ensure it conforms to a known schema or structure, mitigating any unexpected or malicious data.

References:

- [OWASP XML External Entity \(XXE\) Prevention Cheat Sheet](#)

Rec3. Use TLS for communications and protect stored data with encryption

C-WEB-APPLICATION-SERVER-SIDE-CNT-05

Very high

Not tested

State: Recommended

Description:
To secure data during transmission and at rest, ensure that all communications are protected using Transport Layer Security (TLS), and that stored data is encrypted using strong encryption algorithms. Additionally, implement robust key management practices to safeguard encryption keys and ensure that only authorized users and systems can access sensitive data.
Implementation Steps:

- Enforce TLS for All Communications:** Configure all communication channels, including APIs, web traffic, and data transfers, to use TLS (preferably TLS 1.2 or higher) to protect data in transit.
- Encrypt Stored Data:** Use strong encryption algorithms (e.g., AES-256) to encrypt sensitive data at rest, ensuring that unauthorized users cannot access or manipulate it.
- Implement Key Management Practices:** Use a centralized key management system (KMS) to securely generate, store, and rotate encryption keys. Enforce access controls to ensure that only authorized systems can access encryption keys.
- Regularly Update and Rotate Keys:** Set up automated key rotation policies to periodically update encryption keys and minimize the risk of key compromise.

References:

- [OWASP Transport Layer Security \(TLS\) Cheat Sheet](#)

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Check user roles and permissions on every request

C-WEB-APPLICATION-SERVER-SIDE-CNT-03

High

Rec2. Configure XML parsers to disallow external entities and DTD processing

C-WEB-APPLICATION-SERVER-SIDE-CNT-10

Very high

Rec3. Use TLS for communications and protect stored data with encryption

C-WEB-APPLICATION-SERVER-SIDE-CNT-05

Very high

PCI-DSS-v3.2.1: 1.1

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

2 Recommended countermeasures

Rec1. Check user roles and permissions on every request

C-WEB-APPLICATION-SERVER-SIDE-CNT-03

High

Not tested

State: Recommended

Description:
To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.

Report: CodeSentinel Thread Modeling

Compliance report - 2025-03-14T18:54:57.395420405Z

89 of 97

Implementation Steps:

- 1. **Role-Based Access Control (RBAC):** Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.
- 2. **Permission Validation:** For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.
- 3. **Enforce Fine-Grained Access Control:** Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.
- 4. **Session Management:** Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.
- 5. **Audit Logs:** Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.

References:

- [OWASP Access Control Cheat Sheet](#)

Rec2. Block or validate all outbound requests C-WEB-APPLICATION-SERVER-SIDE-CNT-11 Medium Not tested

• State: Recommended

Description:

To reduce the risk of data exfiltration or unauthorized system communication, block or carefully validate all outbound requests made by the application. Ensure that only requests to specific, trusted domains and IP addresses are allowed, and that any outbound request to unapproved destinations is blocked or flagged for further review. This control prevents the application from communicating with malicious external systems and protects sensitive data from being leaked.

Implementation Steps:

- 1. **Implement Outbound Request Filtering:** Use a web application firewall (WAF) or proxy to block or monitor any outbound requests that are not to trusted domains or IP addresses.
- 2. **Validate Outbound Requests:** Before allowing outbound communication, validate the destination domain and IP address to ensure they are within an approved list of resources necessary for the application.
- 3. **Set Domain and IP Allowlists:** Define an allowlist of specific domains and IP addresses that the application is authorized to interact with, and block all other outbound traffic.
- 4. **Monitor and Log Outbound Requests:** Continuously monitor and log all outbound requests for suspicious activity or attempts to communicate with unapproved destinations.
- 5. **Review and Update Allowlist Regularly:** Periodically review the allowlist to ensure that it remains up-to-date with the application's legitimate requirements.

References:

- [Security and Privacy Controls for Information Systems and Organizations](#)

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High

Rec2. Block or validate all outbound requests C-WEB-APPLICATION-SERVER-SIDE-CNT-11 Medium

PCI-DSS-v3.2.1: 1.3

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

2 Recommended countermeasures

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High Not tested

• State: Recommended

Description:

To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.

Implementation Steps:

- 1. **Role-Based Access Control (RBAC):** Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.
- 2. **Permission Validation:** For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.
- 3. **Enforce Fine-Grained Access Control:** Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.
- 4. **Session Management:** Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.
- 5. **Audit Logs:** Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.

References:

- [OWASP Access Control Cheat Sheet](#)

Rec2. Block or validate all outbound requests C-WEB-APPLICATION-SERVER-SIDE-CNT-11 Medium Not tested

• State: Recommended

Description:

To reduce the risk of data exfiltration or unauthorized system communication, block or carefully validate all outbound requests made by the application. Ensure that only requests to specific, trusted domains and IP addresses are allowed, and that any outbound request to unapproved destinations is blocked or flagged for further review. This control prevents the application from communicating with malicious external systems and protects sensitive data from being leaked.

Implementation Steps:

- 1. **Implement Outbound Request Filtering:** Use a web application firewall (WAF) or proxy to block or monitor any outbound requests that are not to trusted domains or IP addresses.
- 2. **Validate Outbound Requests:** Before allowing outbound communication, validate the destination domain and IP address to ensure they are within an approved list of resources necessary for the application.
- 3. **Set Domain and IP Allowlists:** Define an allowlist of specific domains and IP addresses that the application is authorized to interact with, and block all other outbound traffic.
- 4. **Monitor and Log Outbound Requests:** Continuously monitor and log all outbound requests for suspicious activity or attempts to communicate with unapproved destinations.
- 5. **Review and Update Allowlist Regularly:** Periodically review the allowlist to ensure that it remains up-to-date with the application's legitimate requirements.

References:

- [Security and Privacy Controls for Information Systems and Organizations](#)

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

- Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High
- Rec2. Block or validate all outbound requests C-WEB-APPLICATION-SERVER-SIDE-CNT-11 Medium

PCI-DSS-v3.2.1: 1.5

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

1 Recommended countermeasures

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High Not tested

State: Recommended

Description:
To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.
Implementation Steps:

1. Role-Based Access Control (RBAC):

Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.

2. Permission Validation:

For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.

3. Enforce Fine-Grained Access Control:

Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.

4. Session Management:

Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.

5. Audit Logs:

Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.

References:

OWASP Access Control Cheat Sheet

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High

PCI-DSS-v3.2.1: 2.5

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

1 Recommended countermeasures

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High Not tested

State: Recommended

Description:
To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.
Implementation Steps:

1. Role-Based Access Control (RBAC):

Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.

2. Permission Validation:

For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.

3. Enforce Fine-Grained Access Control:

Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.

4. Session Management:

Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.

5. Audit Logs:

Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.

References:

OWASP Access Control Cheat Sheet

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High

PCI-DSS-v3.2.1: 3.7

Report: CodeSentinel Thread Modeling

Compliance report - 2025-03-14T18:54:57.395420405Z

91 of 97

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

1 Recommended countermeasures

Rec1. Check user roles and permissions on every request

C-WEB-APPLICATION-SERVER-SIDE-CNT-03

High

Not tested

State: Recommended

Description:

To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.

Implementation Steps:

- Role-Based Access Control (RBAC):** Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.
- Permission Validation:** For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.
- Enforce Fine-Grained Access Control:** Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.
- Session Management:** Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.
- Audit Logs:** Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.

References:

- [OWASP Access Control Cheat Sheet](#)

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Check user roles and permissions on every request

C-WEB-APPLICATION-SERVER-SIDE-CNT-03

High

PCI-DSS-v3.2.1: 4.3

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

1 Recommended countermeasures

Rec1. Check user roles and permissions on every request

C-WEB-APPLICATION-SERVER-SIDE-CNT-03

High

Not tested

State: Recommended

Description:

To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.

Implementation Steps:

- Role-Based Access Control (RBAC):** Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.
- Permission Validation:** For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.
- Enforce Fine-Grained Access Control:** Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.
- Session Management:** Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.
- Audit Logs:** Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.

References:

- [OWASP Access Control Cheat Sheet](#)

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Check user roles and permissions on every request

C-WEB-APPLICATION-SERVER-SIDE-CNT-03

High

PCI-DSS-v3.2.1: 5.4

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

1 Recommended countermeasures

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High Not tested

- State: Recommended
- Description:

To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.
- Implementation Steps:
 - Role-Based Access Control (RBAC):** Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.
 - Permission Validation:** For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.
 - Enforce Fine-Grained Access Control:** Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.
 - Session Management:** Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.
 - Audit Logs:** Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.
- References:
 - [OWASP Access Control Cheat Sheet](#)

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High

PCI-DSS-v3.2.1: 6.7

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

1 Recommended countermeasures

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High Not tested

- State: Recommended
- Description:

To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.
- Implementation Steps:
 - Role-Based Access Control (RBAC):** Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.
 - Permission Validation:** For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.
 - Enforce Fine-Grained Access Control:** Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.
 - Session Management:** Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.
 - Audit Logs:** Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.
- References:
 - [OWASP Access Control Cheat Sheet](#)

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High

PCI-DSS-v3.2.1: 7.3

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

1 Recommended countermeasures

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High Not tested

- State: Recommended
- Description:

To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.
- Implementation Steps:

1. **Role-Based Access Control (RBAC):** Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.
2. **Permission Validation:** For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.
3. **Enforce Fine-Grained Access Control:** Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.
4. **Session Management:** Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.
5. **Audit Logs:** Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.

References:

- [OWASP Access Control Cheat Sheet](#)

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High

PCI-DSS-v3.2.1: 8.8

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

1 Recommended countermeasures

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High Not tested

- State: Recommended
- Description:

To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.

Implementation Steps:

1. **Role-Based Access Control (RBAC):** Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.
2. **Permission Validation:** For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.
3. **Enforce Fine-Grained Access Control:** Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.
4. **Session Management:** Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.
5. **Audit Logs:** Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.

References:

- [OWASP Access Control Cheat Sheet](#)

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High

PCI-DSS-v3.2.1: 9.10

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

1 Recommended countermeasures

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High Not tested

- State: Recommended
- Description:

To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.

Implementation Steps:

1. **Role-Based Access Control (RBAC):** Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.
2. **Permission Validation:** For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.
3. **Enforce Fine-Grained Access Control:** Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.
4. **Session Management:** Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.
5. **Audit Logs:** Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.

References:

- [OWASP Access Control Cheat Sheet](#)

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High

PCI-DSS-v3.2.1: 11.6

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

1 Recommended countermeasures

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High Not tested

State: Recommended

Description:
To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.

Implementation Steps:

- Role-Based Access Control (RBAC):** Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.
- Permission Validation:** For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.
- Enforce Fine-Grained Access Control:** Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.
- Session Management:** Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.
- Audit Logs:** Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.

References:

- [OWASP Access Control Cheat Sheet](#)

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High

PCI-DSS-v3.2.1: 12.1

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

1 Recommended countermeasures

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High Not tested

State: Recommended

Description:
To ensure secure access control, check the user's roles and permissions on every request for protected resources or functions. This prevents unauthorized access by verifying that the user has the correct privileges to perform the requested action, helping to enforce the principle of least privilege across the application. This should be done dynamically on each request to minimize the risk of privilege escalation or unauthorized resource access.

Implementation Steps:

- Role-Based Access Control (RBAC):** Implement RBAC to define user roles and assign permissions based on the user's role. Ensure that only authorized roles can access specific resources or functions.
- Permission Validation:** For every incoming request, validate that the user's assigned roles have the necessary permissions to access or modify the requested resource or function.
- Enforce Fine-Grained Access Control:** Implement fine-grained permission checks for sensitive actions and resources, ensuring that access is granted based on the exact permissions needed.
- Session Management:** Use session or token-based management to persist user identity and permissions, and ensure that permissions are checked against current session data on each request.
- Audit Logs:** Record and review access attempts, especially for sensitive resources, to detect potential unauthorized access or misuse of roles and permissions.

References:

- [OWASP Access Control Cheat Sheet](#)

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Check user roles and permissions on every request C-WEB-APPLICATION-SERVER-SIDE-CNT-03 High

PCI-DSS-v3.2.1: 6.4.1

Non-compliant countermeasures

"Non-compliant" are those countermeasures that are recommended ("Rec"), rejected ("Rej") or not applicable ("N/A") in the model, but required by the standard. Required ("ReqF") countermeasures which tests have failed are also shown as non-compliant ones.

Component: Web Application - Server side

1 Recommended countermeasures

Rec1. Configure XML parsers to disallow external entities and DTD processing

C-WEB-APPLICATION-SERVER-SIDE-CNT-10

Very high

☒ Recommended

☐ Not tested

State:

☒ Recommended

Description:

To prevent XML-based attacks such as XML External Entity (XXE) and Denial of Service (DoS) attacks, configure XML parsers to disallow external entities and Document Type Definition (DTD) processing by default. This ensures that any incoming XML documents cannot trigger external requests or load external data that could compromise the system, leak sensitive data, or cause resource exhaustion.

Implementation Steps:

1. Disable External Entity Resolution:

Configure XML parsers to explicitly disable the resolution of external entities by setting options or flags that prevent the parser from fetching external resources.

2. Disable DTD Processing:

Ensure that the XML parser does not process DTDs, which can be used to define and reference external entities or cause denial-of-service attacks via large or nested DTDs.

3. Use Secure Parsers:

Ensure that you are using updated and secure XML parsing libraries that offer built-in protections against XXE and other XML-based vulnerabilities.

4. Validate XML Input:

Before parsing, validate XML input to ensure it conforms to a known schema or structure, mitigating any unexpected or malicious data.

References:

[OWASP XML External Entity \(XXE\) Prevention Cheat Sheet](#)

Recommended countermeasures

Below are the recommended countermeasures ("Rec") by component and threat for standard reference.

Component: Web Application - Server side

Rec1. Configure XML parsers to disallow external entities and DTD processing C-WEB-APPLICATION-SERVER-SIDE-CNT-10 Very high

End of Compliance report