BE SOFTWARE ARCHITECTURE WISE · KAPITEL 8

Sicherheit, Datenschutz und Compliance by Design

Sicherheit und Datenschutz sind Eigenschaften der Struktur eines Systems: Sie entscheiden sich daran, wo Vertrauensgrenzen verlaufen, wer was tun darf und wohin Daten fließen. Dieses Kapitel behandelt die Prinzipien sicheren Designs, Bedrohungsmodellierung mit Datenflussdiagrammen, STRIDE und Angriffsbäumen, Identität und Zugriff mit OAuth 2.0 und OpenID Connect, Verschlüsselung, Schlüsselverwaltung und Secrets, Zero Trust zwischen Services, die Sicherheit der Software-Lieferkette, Privacy by Design nach der DSGVO und Sicherheit im Betrieb. Es richtet sich an Entwicklungsteams und Architekten, die die verteilten Architekturen und Datenarchitekturen aus Kapitel 6 und 7 kennen und sie nun sicher machen müssen.

bewiseapps.com/de/software-architecture-wise/security-by-design/ · Version 2026-11-03. Nur Bildungsinformation, keine professionelle Beratung.

AUF EINEN BLICK

Das lernen Sie

- Die CIA-Triade, die Designprinzipien von Saltzer und Schroeder, Defense in Depth und sichere Standardeinstellungen auf Architekturentscheidungen anwenden und erklären, wofür Architekten in einem sicheren Entwicklungslebenszyklus verantwortlich sind.
- Ein Bedrohungsmodell für einen Entwurf erstellen, mit einem Datenflussdiagramm mit Vertrauensgrenzen, STRIDE und Angriffsbäumen, und die daraus folgenden Risiken und Gegenmaßnahmen bewerten und festhalten.
- Authentifizierung von Autorisierung unterscheiden, zwischen Sessions und Tokens wählen und OAuth 2.0 mit PKCE, Client Credentials, OpenID Connect sowie RBAC, ABAC und beziehungsbasierte Zugriffskontrolle erklären.
- Entscheiden, was wo und mit welchen Schlüsseln verschlüsselt wird, mit Envelope Encryption, einem KMS, Passwort-Hashing, Secrets-Management und Tokenisierung.
- Zero-Trust-Architektur erklären und die Sicherheit zwischen Services mit Mutual TLS, Workload Identity und Policy Enforcement Points entwerfen.
- Risiken der Software-Lieferkette mit SBOMs, dem SLSA-Framework, Signaturen, Pinning und isolierten Builds verringern und die Pflichten aus der Cyberresilienz-Verordnung der EU zusammenfassen.
- Für die Grundsätze der DSGVO, den Datenschutz durch Technikgestaltung und durch datenschutzfreundliche Voreinstellungen und die Rechte der betroffenen Personen entwerfen, einschließlich der Löschung über Backups, Caches und Event-Logs hinweg.
- Sicherheits-Logging, Erkennung, Schwachstellenmanagement und Incident Response planen, einschließlich der 72-Stunden-Frist der DSGVO für die Meldung von Verletzungen des Schutzes personenbezogener Daten.

Bevor Sie beginnen

Kapitel 6 und 7, besonders Microservices, ereignisgesteuerte Architektur und Datenhoheit; der Abschnitt über Sicherheit als Qualitätsmerkmal in Kapitel 2; der Abschnitt über Netzwerke, HTTPS und TLS in Kapitel 0.

Dauer

Etwa 4 Stunden Lesezeit; 45 Minuten für den Test.

Schlüsselbegriffe

Vertrauensgrenze · Least Privilege · Defense in Depth · Bedrohungsmodellierung · STRIDE · OAuth 2.0 · OpenID Connect · PKCE · Envelope Encryption · Zero Trust · Mutual TLS · SBOM · Pseudonymisierung · Crypto-Shredding

Sicherheit ist eine Eigenschaft der Struktur

Die meisten schweren Sicherheitsvorfälle entstehen nicht durch einen exotischen Angriff auf die Kryptografie. Sie entstehen aus strukturellen Entscheidungen: ein interner Service, der jedem Aufrufer im Netzwerk vertraute, eine Administrator-API, die eine Autorisierungsprüfung ausließ, ein Datenbank-Backup, das in einen für jeden lesbaren Storage-Bucket kopiert wurde, ein Zugriffsschlüssel, der in ein Repository committet wurde, eine Abhängigkeit, von der niemand wusste, dass das System sie nutzt. Jede davon war eine Architekturentscheidung, meist implizit getroffen, darüber, wo Vertrauen liegt, welche Komponente was tun darf und wohin Daten fließen. Deshalb behandelt dieses Kapitel Sicherheit als Qualitätsmerkmal, das in die Struktur hineinentworfen wird, im Sinne der Sicherheitsszenarien aus Kapitel 2, und nicht als Liste von Produkten, die man am Ende kauft.

Deshalb ist Sicherheit auch untrennbar mit den Stilen aus Kapitel 6 und 7 verbunden. Jede Service-Grenze, jede Queue, jeder Cache, jede Replik und jede Analysekopie vergrößert die Angriffsfläche und schafft einen weiteren Ort, an dem Daten geschützt werden müssen. Verteilung vervielfacht die Zahl der Vertrauensgrenzen; die Datenarchitektur entscheidet, wie viele Kopien personenbezogener Daten existieren und wie lange sie bestehen. Das durchgehende Beispiel ist die Essenslieferplattform, die im ganzen Kurs verwendet wird: Kunden, Restaurants und Kuriere, ein API-Gateway, Services für Bestellungen, Zahlungen, Disposition und Benachrichtigungen, ein Event-Stream und ein Data Warehouse.

Wie die acht Abschnitte zusammenpassen

Die ersten vier Abschnitte bilden den Kern sicheren Designs. **Sicherheitsprinzipien** legt das Vokabular fest: die CIA-Triade, die Designprinzipien von Saltzer und Schroeder aus dem Jahr 1975 wie Least Privilege und ausfallsichere Standardwerte, Defense in Depth, sichere Voreinstellungen, die OWASP Top 10 und einen sicheren Entwicklungslebenszyklus auf Grundlage des SSDF von NIST. **Bedrohungsmodellierung** macht aus diesen Prinzipien eine wiederholbare Designtätigkeit: ein Datenflussdiagramm mit seinen Vertrauensgrenzen zeichnen, mit STRIDE und Angriffsbäumen fragen, was schiefgehen kann, die Risiken bewerten und über Gegenmaßnahmen entscheiden. **Identität und Zugriff** behandelt, wer aufruft: Passwörter nach den aktuellen Empfehlungen von NIST, Passkeys, Sessions und Tokens, OAuth 2.0 mit PKCE, OpenID Connect und die Autorisierungsmodelle RBAC, ABAC und beziehungsbasierte Zugriffskontrolle. **Daten schützen** behandelt Verschlüsselung bei der Übertragung und im Ruhezustand, Passwort-Hashing, Schlüsselverwaltung mit Envelope Encryption, Secrets, Tokenisierung und die Entscheidung, was wo verschlüsselt wird.

Die zweiten vier Abschnitte weiten den Blick. **Zero Trust** ersetzt die Vorstellung eines vertrauenswürdigen internen Netzwerks durch Authentifizierung und Autorisierung bei jeder Anfrage zwischen Services, mit Mutual TLS, Zertifikaten sowie Policy Decision Points und

Policy Enforcement Points. **Sicherheit der Software-Lieferkette** behandelt Abhängigkeiten, Build-Systeme und Container-Images als Teil der Angriffsfläche, mit Software Bills of Materials, dem SLSA-Framework, Signaturen und der Cyberresilienz-Verordnung der EU. **Privacy by Design** erklärt, wie die DSGVO und die UK GDPR die Architektur prägen: Datenminimierung, Pseudonymisierung, die Rechte der betroffenen Personen, Aufbewahrung und Löschung über Backups und Event-Logs hinweg. **Sicherheit im Betrieb** schließt den Kreis mit Sicherheits-Logging, Erkennung, Schwachstellenmanagement, Incident Response und der Meldung von Sicherheitsverletzungen.

Was Architekten entscheiden und was sie delegieren

Ein Architekt muss weder Penetrationstester noch Kryptograf sein und sollte keine eigene Kryptografie erfinden. Seine Aufgabe ist es, die Entscheidungen zu treffen, die teuer zu ändern sind und die kein einzelnes Team allein treffen kann: wo die Vertrauensgrenzen verlaufen, welchen Identity Provider und welches Token-Modell die ganze Plattform nutzt, welche Komponente der Policy Decision Point für welche Ressourcen ist, welche Daten als vertraulich eingestuft und deshalb mit welchen Schlüsseln verschlüsselt werden, wie Services einander ihre Identität nachweisen und wie die Organisation einen Angriff erkennt und darauf reagiert. Diese Entscheidungen gehören in Architecture Decision Records und in die querschnittlichen Konzepte einer arc42-Beschreibung, wie jede andere wichtige Entscheidung.

Entwicklungsteams verantworten andere, ebenso wichtige Anliegen: Eingaben validieren, Ausgaben kodieren, die Authentifizierungs- und Autorisierungsbibliotheken der Plattform richtig nutzen, Fehler behandeln, ohne in einen offenen Zustand zu fallen (Fail-open), Abhängigkeiten aktuell halten und Tests für das Sicherheitsverhalten schreiben. Beide treffen sich in der Bedrohungsmodellierung, in Reviews des sicheren Designs und in automatisierten Prüfungen in der Delivery-Pipeline. Wo Fachleute uneins sind, etwa darüber, wie viel Sicherheitsarbeit auf die Entwicklungsteams verlagert werden sollte oder wie formal Bedrohungsmodellierung sein sollte, stellt das Kapitel beide Positionen dar.

Recht, Normen und Genauigkeit

Sicherheit und Datenschutz sind voller nummerierter Normen und Rechtsvorschriften, und dieses Kapitel benennt sie präzise: NIST Special Publications, IETF-RFCs, OWASP-Projekte, PCI DSS, die DSGVO und die UK GDPR sowie die Cyberresilienz-Verordnung der EU. Normen ändern sich, deshalb nennt jede Erwähnung die zum Zeitpunkt der im Abschnitt angegebenen letzten Prüfung aktuelle Fassung und verweist auf die maßgebliche Quelle. Die Ausführungen zur Regulierung erklären, wie sich das Recht auf die Architektur auswirkt; sie sind Information, keine Rechtsberatung, und für eine konkrete Situation bleiben die Juristen und der Datenschutzbeauftragte der eigenen Organisation die richtigen Ansprechpartner.

Der eine Gedanke zum Mitnehmen

Entscheiden Sie bewusst, früh und schriftlich, wem und was Ihr System vertraut, wofür und warum, und machen Sie jede dieser Vertrauensentscheidungen überprüfbar. Die meisten Techniken in diesem Kapitel, von Bedrohungsmodellen und Least Privilege bis zu OAuth-Scopes, Envelope Encryption, Mutual TLS, signierten Builds und Audit-Logs, sind Wege, implizites Vertrauen durch explizites, überprüfbares Vertrauen zu ersetzen. Ein so entworfenes System begrenzt den Schaden des Sicherheitsvorfalls, der irgendwann kommen wird, und kann seinen Kunden, Prüfern und Aufsichtsbehörden zeigen, wie es sie schützt.

Sicherheitsprinzipien für Architekten

BEVOR SIE DAS LESEN

Dieser Abschnitt baut auf:

- 0.4 Wie Computer kommunizieren
- 0.6 APIs und Integration
- 2.7 Sicherheit, Benutzbarkeit und mehr

Begriffe von früher, die er verwendet: CIA-Triade, API-Gateway, TLS und HTTPS, Rate Limit

DIE IDEE IN KÜRZE

Denken Sie daran, wie eine Bankfiliale geschützt ist. Der Tresor wird nicht von einem einzigen Schloss bewacht, sondern von mehreren: einer verschlossenen Eingangstür, Personal, das nur Schlüssel für die Räume hat, die es braucht, einem Tresor, den nur zwei Personen gemeinsam öffnen können, Kameras und einer Alarmanlage, die die Polizei ruft. Das Gebäude wurde von Anfang an darauf ausgelegt; niemand hat nachträglich einen Tresor an einen Laden angeschraubt. Softwaresicherheit funktioniert genauso. Architekten entscheiden früh, wo die verschlossenen Türen sind, wer welche Schlüssel bekommt und was passiert, wenn ein Schloss versagt, damit ein einzelner Fehler nicht alles öffnet.

WAS SIE WISSEN MÜSSEN

- Sicherheit ist eine Qualität der Struktur: Wo Vertrauensgrenzen verlaufen, welche Komponente was tun darf und wohin Daten fließen, entscheidet über die meisten Sicherheitseigenschaften, und die meisten Sicherheitsvorfälle nutzen solche Entscheidungen aus, nicht die Kryptografie.
- Vertraulichkeit, Integrität und Verfügbarkeit sowie Authentizität, Verantwortlichkeit und Nichtabstreitbarkeit müssen je Asset benannt werden, weil sie in Konflikt geraten: Fail-closed schützt Daten, kostet aber Verfügbarkeit.
- Die Prinzipien von Saltzer und Schroeder aus dem Jahr 1975 (Sparsamkeit der Mechanismen, ausfallsichere Standardwerte, vollständige Zugriffsprüfung, offenes Design, Trennung von Privilegien, Least Privilege, möglichst wenige gemeinsame Mechanismen, psychologische Akzeptanz) bleiben die Grundlage sicheren Designs.
- Defense in Depth schichtet unabhängige Kontrollen, sodass ein einzelnes Versagen kein Asset offenlegt; sichere Voreinstellungen und eine Paved Road machen den einfachen Weg zum sicheren.

- Verkleinern Sie die Angriffsfläche und setzen Sie Authentifizierung, Autorisierung und Validierung an jede Vertrauensgrenze; alles auf dem Client liegt außerhalb der Grenze.
- Die OWASP Top 10:2025 sind eine Sensibilisierungsliste, keine Norm; das SSDF von NIST (SP 800-218) beschreibt einen sicheren Entwicklungslebenszyklus, und Architekten verantworten die teamübergreifenden Sicherheitsentscheidungen, während die Teams für deren korrekte Anwendung verantwortlich sind.

Sicherheit als Qualität der Struktur

Kapitel 2 hat Sicherheit als Qualitätsmerkmal eingeführt: die Fähigkeit des Systems, Daten und Funktionen vor unbefugter Nutzung zu schützen und zugleich den Befugten zu dienen, ausgedrückt durch testbare Sicherheitsszenarien, die einen Angreifer, einen Angriff, ein Ziel und eine messbare Reaktion benennen. Dieses Kapitel geht den nächsten Schritt und fragt, wie Architektur diese Qualität hervorbringt. Die Antwort lautet: Die meisten Sicherheitseigenschaften werden durch die Struktur entschieden, also dadurch, welche Komponenten es gibt, welche davon welchen anderen vertrauen, wo Daten gespeichert und kopiert werden und durch welche wenigen Punkte jede Anfrage laufen muss.

Betrachten Sie, wie reale Sicherheitsvorfälle entstehen. Ein Angreifer ändert die Bestellnummer in einer URL und sieht die Adresse eines anderen Kunden, weil der Service geprüft hat, ob der Benutzer angemeldet ist, aber nicht, ob die Bestellung ihm gehört. Ein Storage-Bucket mit Datenbankexporten ist aus dem Internet lesbar, weil vor Jahren eine Voreinstellung geändert wurde. Ein interner Reporting-Service akzeptiert jede Anfrage aus dem Netzwerk, sodass ein einziger kompromittierter Laptop jeden Kundendatensatz erreicht. Keiner dieser Fälle braucht raffinierte Kryptografie, um ausgenutzt zu werden; jeder ist eine Designentscheidung über Vertrauen, die niemand ausdrücklich getroffen hat. Security by Design heißt, diese Entscheidungen bewusst, früh und schriftlich zu treffen und es dann schwer zu machen, sie versehentlich zu verletzen.

Sicherheitsarchitektur steht deshalb neben den anderen Querschnittsanliegen aus Abschnitt 8 von arc42: ein Identitätsmodell, ein Autorisierungsmodell, ein Datenschutzkonzept und ein Logging-Konzept, an die sich jedes Team hält. Der Rest dieses Abschnitts legt das Vokabular und die Prinzipien fest, auf denen die übrigen Abschnitte des Kapitels aufbauen.

Vertraulichkeit, Integrität und Verfügbarkeit in einem realen System

Die CIA-Triade ist der Ausgangspunkt. Auf der Essenslieferplattform umfasst **Vertraulichkeit** die Namen, Adressen, Telefonnummern und Bestellhistorien der Kunden, die Live-Standorte und Ausweisdokumente der Kuriere, die Bankverbindungen der Restaurants für Auszahlungen und die eigenen Secrets der Plattform. **Integrität** umfasst Preise, Warenkorbsummen,

Erstattungen, Kurierauszahlungen und die Speisekarte, die ein Kunde sieht: Ein Angreifer, der den Preis eines Gerichts auf null setzen oder die Auszahlungen eines Restaurants auf sein eigenes Bankkonto umleiten kann, hat nichts Geheimes gelesen, aber ernsten Schaden angerichtet. **Verfügbarkeit** zählt am meisten während der Abendspitze, wenn ein Denial-of-Service-Angriff oder eine Ransomware-Infektion jede Minute Umsatz kostet.

ISO/IEC 25010 nennt außerdem **Authentizität** (Identitäten lassen sich nachweisen), **Verantwortlichkeit** (Aktionen lassen sich auf den zurückführen, der sie ausgeführt hat), **Nichtabstreitbarkeit** (es lässt sich nachweisen, dass Aktionen stattgefunden haben) und, neu in der Ausgabe von 2023, **Widerstandsfähigkeit** (das Produkt arbeitet auch unter Angriff wie vorgesehen weiter). Architekten sollten für jedes Asset benennen, welche Eigenschaften zählen, weil sie in verschiedene Richtungen ziehen. Fail-closed schützt Vertraulichkeit und Integrität, kostet aber Verfügbarkeit: Ist der Autorisierungsservice ausgefallen, lehnt ein Fail-closed-Bestellservice Bestellungen ab. Fail-open bewahrt die Verfügbarkeit und setzt alles andere aufs Spiel. Die richtige Wahl unterscheidet sich je Vorgang: Die Anzeige einer öffentlichen Speisekarte kann per Fail-open auf eine zwischengespeicherte Kopie ausweichen, eine Erstattung dagegen muss fail-closed sein.

Die Designprinzipien von Saltzer und Schroeder

1975 veröffentlichten Jerome Saltzer und Michael Schroeder, gestützt auf ihre Erfahrung mit dem Betriebssystem Multics am MIT, in *The Protection of Information in Computer Systems* acht Designprinzipien für Schutzmechanismen. Ein halbes Jahrhundert später sind sie noch immer die meistzitierte Grundlage sicheren Designs, weil es in ihnen um Struktur geht und nicht um Technologie. Die Tabelle formuliert jedes Prinzip neu und gibt ein modernes Beispiel von der Essenslieferplattform.

Die acht Designprinzipien mit modernen Beispielen

Prinzip	Was es verlangt	Modernes Beispiel
Sparsamkeit der Mechanismen	Das Sicherheitsdesign so klein und einfach wie möglich halten, damit es gründlich geprüft und getestet werden kann.	Eine gemeinsame, gut geprüfte Autorisierungsbibliothek oder Policy Engine statt Berechtigungsprüfungen, die in jedem Service anders von Hand geschrieben sind.
Ausfallsichere Standardwerte	Zugriff auf ausdrückliche Erlaubnis stützen; im Zweifel oder wenn etwas ausfällt, lautet die Antwort Nein.	Ein neuer API-Endpoint bleibt gesperrt, bis eine Policy den Zugriff gewährt; eine nicht erreichbare Policy Engine bedeutet Ablehnen, nicht Erlauben.
Vollständige Zugriffsprüfung	Jeden Zugriff auf jedes Objekt jedes Mal prüfen, nicht nur beim ersten Mal.	Der Bestellservice prüft bei jedem `GET /orders/{id}` die Eigentümerschaft, statt jeder ID von einem angemeldeten Benutzer zu vertrauen; zwischengespeicherte Entscheidungen verfallen schnell.
Offenes Design	Sicherheit darf nicht davon abhängen, dass das Design geheim ist; geheim sein sollten nur Schlüssel und Passwörter.	Veröffentlichte Algorithmen (AES, TLS 1.3) und Standardprotokolle (OAuth 2.0), die sicher bleiben, auch wenn Angreifer den Quellcode lesen.
Trennung von Privilegien	Wo möglich, zwei unabhängige Bedingungen (zwei Schlüssel) verlangen, bevor Zugriff gewährt wird.	Erstattungen über einem Schwellenwert brauchen eine zweite Freigabe; ein Produktions-Release braucht sowohl ein Code-Review als auch eine erfolgreiche Pipeline; Anmeldung mit mehreren Faktoren.
Least Privilege	Jedem Programm und jedem Benutzer nur die Rechte geben, die für die Aufgabe nötig sind, und nur so lange wie nötig.	Das Datenbankkonto des Speisekarten-Service kann das Zahlungsschema nicht lesen; die Cloud-Rolle eines Build-Jobs kann Images pushen, aber keine Cluster löschen.
Möglichst wenige gemeinsame Mechanismen	Mechanismen, die Benutzer gemeinsam nutzen, minimieren, weil jeder gemeinsame Mechanismus ein Weg ist, über den Informationen abfließen können.	Getrennte Caches, Queues oder Connection-Pools je Mandant oder Vertrauensstufe; ein gemeinsamer Cache mit unachtsam gewählten Cache-Keys liefert die Daten eines Benutzers an einen anderen aus.
Psychologische Akzeptanz	Schutz muss leicht richtig zu benutzen sein, sonst umgehen ihn die Menschen.	Passkeys statt komplizierter Passwortregeln; sichere Voreinstellungen in der Plattform; Warnungen, die selten sind und deshalb gelesen werden.

WOHER DAS KOMMT

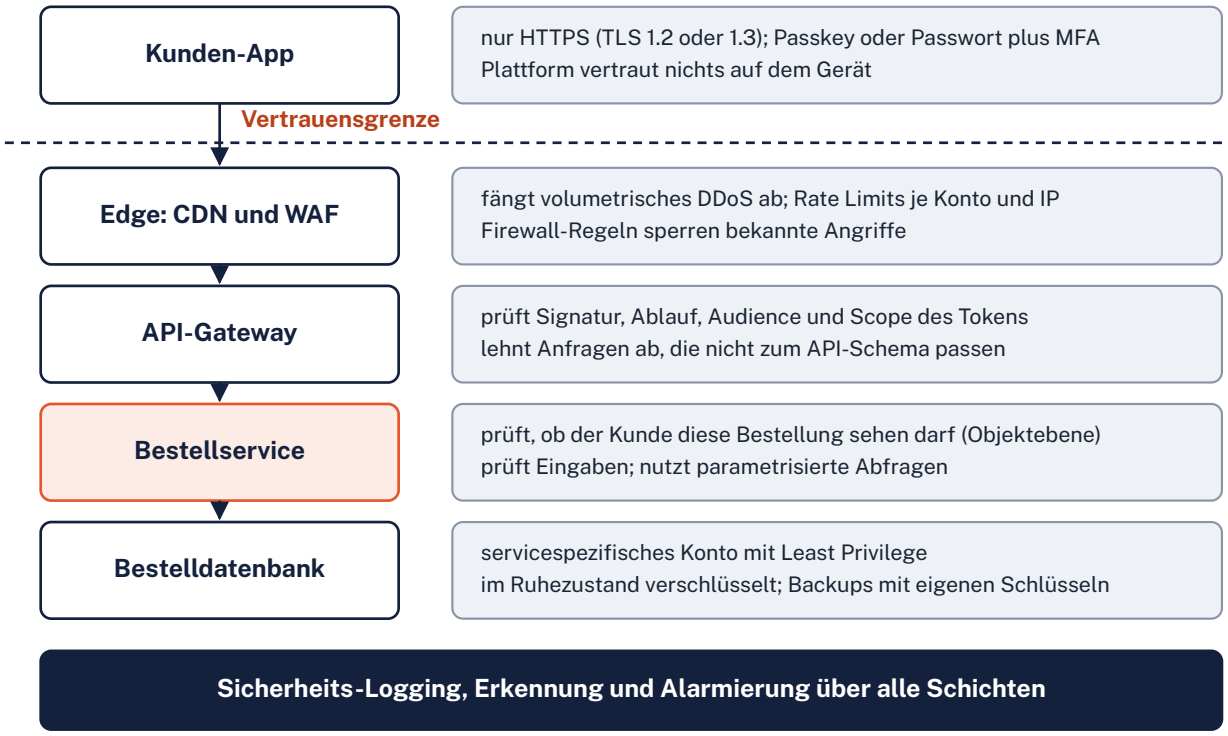
Woher die Prinzipien kommen

Der Aufsatz von Saltzer und Schroeder erschien im September 1975 in den *Proceedings of the IEEE*. Neben den acht Prinzipien beschrieb er zwei weitere, **Arbeitsaufwand** (die Kosten, einen Mechanismus zu umgehen, mit den Mitteln des Angreifers vergleichen) und **Aufzeichnung von Kompromittierungen** (festhalten, dass eine Kompromittierung stattgefunden hat, auch wenn sie sich nicht verhindern lässt), die nach Ansicht der Autoren nur unvollkommen auf Computersysteme passen. Beide leben heute fort: der Arbeitsaufwand im Passwort-Hashing und in Schlüssellängen, die Aufzeichnung von Kompromittierungen in manipulationssicheren Audit-Logs. Offenes Design greift das Prinzip auf, das Auguste Kerckhoffs 1883 für militärische Chiffren formulierte: Ein System soll sicher bleiben, auch wenn alles an ihm außer dem Schlüssel öffentlich ist.

Defense in Depth

Defense in Depth bedeutet, mehrere unabhängige Kontrollen übereinanderzuschichten, sodass das Versagen oder Umgehen einer einzelnen das Asset nicht offenlegt. Die Idee stammt aus dem militärischen Festungsbau und ist in der funktionalen Sicherheit ebenso Standardpraxis wie in der IT-Sicherheit. Ihr Wert hängt ganz am Wort *unabhängig*: Zwei Schichten, die aus demselben Grund versagen, etwa zwei Services, die beide einem gefälschten Header vertrauen, sind eine Schicht, die zweimal gezeichnet wurde.

Das Diagramm zeigt die Schichten auf dem Bestellpfad der Essenslieferplattform. Hat die Token-Validierung des Gateways einen Bug, verhindert die Prüfung auf Objektebene im Bestellservice trotzdem, dass ein Kunde die Bestellung eines anderen liest. Stiehlt ein Angreifer die Disk-Snapshots der Datenbank, schützt sie die Verschlüsselung im Ruhezustand mit Schlüsseln, die anderswo liegen. Versagt all das, sollte das Monitoring das ungewöhnliche Anfragemuster bemerken, und jemand sollte reagieren. Defense in Depth ist auch die ehrliche Antwort auf die Frage „Was, wenn diese Kontrolle versagt?“, die jedes Bedrohungsmodell stellt.



Jede Schicht wendet eigene Kontrollen an; eine einzelne umgangene Kontrolle legt die Daten nicht offen.

WENN DAS NEU FÜR SIE IST

Schritt für Schritt: wenn zwei Schlösser in Wahrheit eines sind

Die Idee, die hier die meiste Sorgfalt verlangt, ist, dass Defense in Depth nur funktioniert, wenn die Schichten unabhängig sind, sodass jede einen Angriff stoppen kann, der an den anderen vorbeikommt. Verfolgen Sie eine Anfrage entlang des Bestellpfads im Diagramm.

- Ein angemeldeter Kunde möchte seine Bestellung 1042 sehen. Die App sendet die Anfrage mit dem Access Token des Kunden, das zeigt, wer er ist.
- Das API-Gateway prüft, ob das Token echt, nicht abgelaufen und für diese API bestimmt ist, und leitet die Anfrage an den Bestellservice weiter.
- Nun ändert der Kunde die Anfrage und ersetzt 1042 durch 1043, die Bestellung eines anderen. Das Token ist weiterhin gültig, also lässt das Gateway die Anfrage durch: Es kann nicht wissen, wem welche Bestellung gehört. Der Bestellservice prüft, stellt fest, dass Bestellung 1043 nicht diesem Kunden gehört, und lehnt ab.
- Diese zweite Prüfung ist eine eigene Schicht. Selbst wenn ein Bug ein ungültiges Token am Gateway vorbeiließe, würde der Bestellservice sich weiterhin weigern, einem Kunden die Bestellung eines anderen zu zeigen.
- Angenommen aber, der Bestellservice prüfte ebenfalls das Token, mit derselben gemeinsamen Bibliothek wie das Gateway, und ein Fehler in dieser Bibliothek akzeptierte Tokens ohne gültige Signatur. Ein einziges gefälschtes Token würde dann beide Prüfungen auf einmal täuschen. Zwei Prüfungen, die aus demselben Grund versagen, sind eine Schicht, die zweimal gezeichnet wurde.
- Die unteren Schichten folgen derselben Regel. Die Verschlüsselung im Ruhezustand schützt gestohlene Disk-Snapshots nur, weil ihre Schlüssel anderswo liegen, und das Monitoring ist dafür da, einen Angriff zu bemerken, wenn alles andere versagt.
- **Die Lehre:** Fragen Sie bei jeder Kontrolle „Was, wenn genau diese versagt?“, und stellen Sie sicher, dass das, was sie überwindet, nicht auch die Schichten dahinter überwindet.

Secure by Default und Security by Design

Ein Produkt ist Secure by Default, wenn seine Konfiguration im Auslieferungszustand die sichere ist, sodass jede Schwächung des Schutzes eine bewusste Handlung erfordert. Viele große Sicherheitsvorfälle gingen auf unsichere Voreinstellungen zurück, die nie geändert wurden: Datenbanken, die ohne Passwort im Internet erreichbar waren, Administratorkonsolen mit Werkszugangsdaten, Storage-Buckets, die sich mit einer einzigen Einstellung öffentlich machen ließen. Die Anbieter haben reagiert: Seit April 2023 werden zum Beispiel neue Amazon-S3-Buckets mit eingeschaltetem Block Public Access und deaktivierten Access

Control Lists angelegt. 2023 veröffentlichten die US-amerikanische Cybersecurity and Infrastructure Security Agency (CISA) und Partnerbehörden aus mehreren Ländern gemeinsame Leitlinien, die Hersteller drängen, Produkte Secure by Design und Secure by Default zu gestalten, und 2024 startete die CISA eine freiwillige Secure-by-Design-Selbstverpflichtung für Softwarehersteller.

Für Architekten gilt Secure by Default für die Plattform ebenso wie für Produkte. Enthält die gemeinsame Service-Vorlage bereits Token-Validierung, strukturiertes Sicherheits-Logging, einen Container-Benutzer ohne Root-Rechte und ein Datenbankkonto mit Least Privilege, startet jeder neue Service sicher, und Teams müssen sich ausdrücklich dagegen entscheiden, was im Review sichtbar wird. Das nennen manche Organisationen eine **Paved Road**: Der einfachste Weg ist zugleich der sichere.

Eng damit verwandt ist die Frage, wie sich ein System verhält, wenn etwas schiefgeht. Die OWASP Top 10:2025 haben eine Kategorie für den fehlerhaften Umgang mit Ausnahmesituationen aufgenommen, zu der auch Fail-open gehört: ein Fehler in einer Berechtigungsprüfung, der abgefangen und ignoriert wird, ein Timeout, das standardmäßig „erlaubt“ ergibt, ein Parsing-Fehler, der die Validierung überspringt. Ausfallsichere Standardwerte, im Code angewendet, bedeuten, dass jeder Ausnahmepfad in einer Sicherheitsentscheidung in einer Ablehnung endet.

Angriffsfläche und Vertrauensgrenzen

Die Angriffsfläche ist alles, was ein Angreifer erreichen und beeinflussen kann: öffentliche Endpoints und ihre Parameter, offene Ports, Datei-Uploads, Message Queues, die Eingaben von Partnern annehmen, Webhooks, Administratorkonsolen, mobile Apps, die sich dekompilieren lassen, sowie der Code von Drittanbietern und die Menschen, denen das System vertraut. Michael Howard bei Microsoft machte Anfang der 2000er-Jahre die Idee bekannt, sie zu messen; die praktische Regel lautet, sie so klein wie möglich zu halten und genau zu wissen, was dazugehört. Einen ungenutzten Endpoint zu entfernen, einen Administratorport zum Internet hin zu schließen oder ein Feature abzuschalten, das niemand nutzt, senkt das Risiko günstiger als jede zusätzliche Kontrolle. Jede Wahl eines Architekturstils in Kapitel 6 hat hier einen Sicherheitspreis: Jeder neue, über das Netzwerk erreichbare Service bringt Endpoints, Zugangsdaten und Datenverkehr hinzu, die geschützt werden müssen.

Eine Vertrauensgrenze ist die Stelle, an der Daten oder Steuerung zwischen Teilen mit unterschiedlichem Vertrauens- oder Berechtigungsniveau wechseln. Die offensichtlichste liegt zwischen dem Internet und der Plattform, aber es gibt viele weitere: zwischen der Kunden-App und der API, zwischen dem Tablet eines Restaurants und dem Speisekarten-Service, zwischen Services verschiedener Teams, zwischen der Plattform und einem Zahlungsanbieter, zwischen Anwendungscode und Datenbank sowie zwischen normalen Benutzern und Administratoren.

Vertrauensgrenzen sind wichtig, weil dort die Prüfungen hingehören: Authentifizierung, Autorisierung, Eingabevalidierung und Ausgabekodierung finden statt, wenn Daten eine Grenze überqueren, nie unter der Annahme, die andere Seite habe es schon erledigt. Die Vertrauensgrenzen in ein Datenflussdiagramm einzuzeichnen, ist der erste Schritt der Bedrohungsmodellierung.

ACHTUNG

Der Client liegt außerhalb der Grenze

Alles, was auf dem Gerät des Kunden läuft, ob mobile App, Single-Page-Webanwendung oder das Tablet eines Restaurants, steht unter der Kontrolle des Angreifers. Preise, Rabatte, Berechtigungen und Bestellsummen, die auf dem Client berechnet werden, sind Vorschläge; der Server muss sie neu berechnen oder überprüfen. Einen Button in der Benutzeroberfläche auszublenden, ist keine Autorisierung. In einer mobilen App eingebettete API-Schlüssel sind nicht geheim, weil sich die App dekompilieren lässt.

Die OWASP Top 10 als Sensibilisierungsliste

Die OWASP Top 10 sind die Rangliste der kritischsten Sicherheitsrisiken für Webanwendungen des Open Worldwide Application Security Project, zusammengestellt aus eingereichten Schwachstellendaten und einer Umfrage in der Community. Die aktuelle Ausgabe, die **OWASP Top 10:2025**, ist die achte. Gegenüber der Ausgabe von 2021 bleibt Broken Access Control auf Platz eins und umfasst nun auch Server-Side Request Forgery, Security Misconfiguration steigt auf Platz zwei, die frühere Kategorie Vulnerable and Outdated Components wird zu **Software Supply Chain Failures** erweitert, und eine neue Kategorie behandelt **Mishandling of Exceptional Conditions**.

OWASP Top 10:2025 und die architektonischen Hebel

Kategorie	Was schiefgeht	Architektonischer Hebel
A01 Broken Access Control	Benutzer handeln außerhalb ihrer Berechtigungen, etwa indem sie durch Ändern einer ID die Bestellung eines anderen Kunden lesen; umfasst Server-Side Request Forgery.	Ein Autorisierungsmodell, standardmäßig ablehnen, Prüfungen auf Objektebene in jedem Service, eingeschränkter ausgehender Netzwerkzugriff.
A02 Security Misconfiguration	Unsichere Voreinstellungen, offener Speicher, ausführliche Fehlerseiten, unnötige Features, die eingeschaltet bleiben.	Infrastructure as Code, gehärtete Vorlagen, automatisierte Policy-Prüfungen in der Pipeline.
A03 Software Supply Chain Failures	Verwundbare oder kompromittierte Abhängigkeiten, Build-Werkzeuge und Vertriebskanäle.	Inventar der Abhängigkeiten (SBOM), Pinning, signierte Herkunftsnachweise; siehe Sicherheit der Software-Lieferkette.
A04 Cryptographic Failures	Unverschlüsselte sensible Daten, schwache Algorithmen, schlechter Umgang mit Schlüsseln.	Datenklassifizierung, TLS überall, ein KMS und Envelope Encryption.
A05 Injection	Nicht vertrauenswürdige Daten werden als Code interpretiert: SQL, Betriebssystembefehle, Cross-Site-Scripting.	Parametrisierte Abfragen, kontextbezogene Ausgabekodierung, sichere Framework-APIs.
A06 Insecure Design	Fehlende oder unwirksame Kontrollen, die sich auch durch noch so korrektes Programmieren nicht beheben lassen.	Bedrohungsmodellierung, sichere Entwurfsmuster, Missbrauchsfälle neben den User Stories.
A07 Authentication Failures	Credential Stuffing, schwache Wiederherstellungsabläufe, Fehler im Session-Handling.	Ein zentraler Identity Provider, MFA oder Passkeys, Rate Limits bei der Anmeldung.
A08 Software or Data Integrity Failures	Code, Updates oder Daten vertrauen, ohne ihre Integrität zu prüfen, etwa unsignierte Updates oder unsichere Deserialisierung.	Signaturprüfung, Integritätsprüfungen für Daten, die Grenzen überqueren.
A09 Security Logging and Alerting Failures	Angriffe werden nicht aufgezeichnet oder zwar aufgezeichnet, aber nie bemerkt oder bearbeitet.	Ein Standard für Sicherheits-Logging, Alarmierung, Runbooks für Incidents.
A10 Mishandling of Exceptional Conditions	Uneinheitlich behandelte Fehler, Systeme mit Fail-open, Fehlermeldungen, die Details preisgeben.	Fail-closed, zentrale Fehlerbehandlung, Ressourcenlimits.

Behandeln Sie die Liste als das, was sie laut OWASP ist: ein Dokument zur Sensibilisierung, keine Norm. Sie ist nützlich für Schulungen, um zu prüfen, ob ein Bedrohungsmodell nicht eine ganze Klasse von Problemen übersehen hat, und für Gespräche mit Stakeholdern. Sie ist keine vollständige Sammlung von Anforderungen, und „Wir sind OWASP-Top-10-konform“ bedeutet wenig. Für überprüfbare Anforderungen listet der Application Security Verification Standard (ASVS) von OWASP testbare Kontrollen auf drei Strengestufen auf, und die separate OWASP API Security Top 10 behandelt API-spezifische Risiken wie Broken Object Level Authorization.

Ein sicherer Entwicklungslebenszyklus und Shift Left

Ein sicherer Entwicklungslebenszyklus baut Sicherheitsaktivitäten in jede Phase der Auslieferung ein, statt sich auf einen Test kurz vor dem Release zu verlassen. Microsoft machte seinen Security Development Lifecycle 2004 verbindlich, nach dem Trustworthy-Computing-Memo von Bill Gates aus dem Jahr 2002, und die Idee verbreitete sich in der ganzen Branche. Die aktuelle Referenz ist das Secure Software Development Framework (SSDF) von NIST, SP 800-218, dessen Version 1.1 im Februar 2022 veröffentlicht wurde; im Dezember 2025 legte NIST einen Entwurf einer Überarbeitung (Version 1.2) zur Kommentierung vor, der zum Zeitpunkt des Schreibens noch nicht final war. Das SSDF gliedert die Praktiken in vier Gruppen: **die Organisation vorbereiten** (Rollen, Schulung, Toolchains, Sicherheitsanforderungen), **die Software schützen** (Code und Build-Integrität schützen, Herkunftsnachweise aufbewahren), **gut gesicherte Software produzieren** (so entwerfen, dass die Sicherheitsanforderungen erfüllt werden, Bedrohungsmodelle erstellen, prüfen und testen, standardmäßig sicher konfigurieren) und **auf Schwachstellen reagieren** (erkennen, beheben und die Grundursachen analysieren, damit dieselbe Klasse nicht wiederkehrt). Es beschreibt Ergebnisse, statt Werkzeuge vorzuschreiben, und ist deshalb eine gute Checkliste für einen Architekten, der einen Auslieferungsprozess bewertet.

Shift Left heißt, Sicherheitsarbeit nach vorn zu verlagern: Sicherheitsanforderungen und Missbrauchsfälle zusammen mit den User Stories, Bedrohungsmodellierung im Design, statische Analyse und Secret-Scanning bei jedem Commit, Abhängigkeitsprüfungen in der Pipeline. Das wirtschaftliche Argument kennen Sie aus Kapitel 1: Eine fehlende Autorisierungsprüfung, die im Design-Review gefunden wird, kostet eine Stunde; derselbe Fehler, nach einem Sicherheitsvorfall gefunden, kostet eine Meldung an die Aufsichtsbehörden. Fachleute sind uneins, wie weit man das treiben sollte. Befürworter sagen, Entwickler, die Sicherheit verstehen, schreiben weniger Schwachstellen. Kritiker weisen darauf hin, dass es Entwicklungsteams mit minderwertigen Befunden überflutet, wenn man ihnen einfach Sicherheitswerkzeuge vorschaltet, und Sicherheit zur Teilzeitaufgabe aller und zur Expertise von niemandem macht; sie plädieren für sichere Voreinstellungen in der Plattform (die Paved Road), damit der sichere Weg keinen Zusatzaufwand erfordert, und für Sicherheitsfachleute, die dort eingebunden sind, wo das Risiko am höchsten ist. Die meisten

reifen Organisationen tun beides und testen und überwachen außerdem weiterhin in der Produktion, weshalb manche inzwischen davon sprechen, Sicherheit überall hin zu verlagern („Shift Everywhere“).

Das laufende System testen: DAST und Fuzzing

Statische Analyse liest Code; diese Techniken führen ihn aus. **Dynamic Application Security Testing** (DAST) testet eine laufende Anwendung von außen mit feindlichen Anfragen, ohne ihren Quellcode zu benötigen, und sucht nach Schwachstellen wie SQL-Injection, Cross-Site-Scripting, schwacher Authentifizierung und Fehlkonfiguration des Servers. Es eignet sich für einen nächtlichen Lauf gegen eine Testumgebung, findet aber nur, was es erreichen kann, und ist bei Geschäftslogik meist schwach, etwa wenn ein Kunde die Bestellung eines anderen ansieht. **Fuzzing** füttert ein Programm mit fehlerhaften oder unerwarteten Eingaben, bis es versagt. Es begann mit der Gruppe von Barton Miller an der University of Wisconsin-Madison, deren Tool `fuzz` etwa ein Viertel bis ein Drittel der getesteten UNIX-Hilfsprogramme zum Absturz oder Hängen brachte. Besonders nützlich ist es für Parser wie den Speisekarten-Importcode der Plattform. Googles OSS-Fuzz betreibt es in großem Maßstab für Open-Source-Projekte.

Was Architekten verantworten und was Entwicklungsteams verantworten

Sicherheitsarbeit ist geteilt, und Unklarheit darüber, wer was verantwortet, ist selbst eine Schwachstelle. Der Architekt verantwortet Entscheidungen, die Teamgrenzen überschreiten und teuer zu ändern sind; die Teams verantworten deren korrekte Anwendung. Ein verbreitetes Muster ist ein kleines zentrales Sicherheitsteam, das Standards setzt und die Sicherheitsdienste der Plattform betreibt, Security Champions in jedem Produktteam, die Bedrohungsmodellierung und Reviews leiten, und Architekten, die dafür sorgen, dass die strukturellen Entscheidungen getroffen und festgehalten werden.

Aufteilung der Sicherheitsverantwortung

Anliegen	Architekt (und Sicherheitsarchitektur)	Entwicklungsteams
Vertrauensgrenzen und Datenflüsse	Sie festlegen, dokumentieren und bei wichtigen Änderungen die Bedrohungsmodellierung leiten.	Den Code mit ihnen im Einklang halten; Änderungen ansprechen, die eine Grenze verschieben.
Identität und Zugriff	Identity Provider, Token-Modell, Autorisierungsmodell und den Ort der Policy-Entscheidung wählen.	Autorisierung an jedem Endpoint und jedem Objekt durchsetzen und testen.
Datenschutz	Daten klassifizieren; entscheiden, was wo mit welchen Schlüsseln verschlüsselt wird; KMS und Secrets-Vault wählen.	Die freigegebenen Bibliotheken nutzen; nie Secrets und nie unnötig personenbezogene Daten loggen.
Umgang mit Ein- und Ausgaben	Strategie für Validierung und Kodierung an Grenzen festlegen; gemeinsame Bibliotheken bereitstellen.	In jedem Handler validieren, kodieren und parametrisieren.
Abhängigkeiten und Build	Pipeline-Kontrollen, freigegebene Quellen und SBOM-Policy festlegen.	Abhängigkeiten aktuell halten; auf Warnungen innerhalb der vereinbarten Fristen reagieren.
Erkennung und Reaktion	Sicherheits-Logging, Audit-Events und Rollen für Incidents festlegen.	Die vereinbarten Events ausgeben; Schwachstellen innerhalb der vereinbarten Service-Levels beheben.

AUSGEARBEITETES BEISPIEL

Ausgearbeitetes Beispiel: die Prinzipien auf eine Erstattungsfunktion für den Support anwenden

Support-Mitarbeitende der Essenslieferplattform müssen Kunden für fehlgeschlagene Lieferungen über das Backoffice-Tool Geld erstatten können. Ein erster Entwurf lässt jeden Mitarbeitenden den Erstattungs-Endpoint des Zahlungsservice mit beliebiger Bestellnummer und beliebigem Betrag aufrufen. Die Tabelle zeigt, wie die Prinzipien den Entwurf umgestalten.

Least Privilege	Mitarbeitende erhalten eine `refund:create`-Berechtigung nur für Bestellungen in ihrem eigenen Markt; die Service-Identität des Backoffice-Tools darf die Erstattungs-API aufrufen, aber nie Kartendaten lesen.
Ausfallsichere Standardwerte	Schlägt die Policy-Prüfung oder das Nachschlagen der Bestellung fehl oder läuft in ein Timeout, wird die Erstattung mit dem Hinweis „Bitte erneut versuchen“ abgelehnt; es gibt keinen Ausweichpfad, der Prüfungen überspringt.
Vollständige Zugriffsprüfung	Der Zahlungsservice selbst prüft jede Erstattung erneut gegen die ursprüngliche Belastung (die Erstattung darf den eingezogenen Betrag abzüglich früherer Erstattungen nicht übersteigen), statt dem Backoffice-Tool zu vertrauen.
Trennung von Privilegien	Erstattungen über £100 oder eine dritte Erstattung an denselben Kunden innerhalb einer Woche brauchen die Freigabe einer Vorgesetzten oder eines Vorgesetzten im selben Tool.
Sparsamkeit der Mechanismen	Nutzt die vorhandene Policy Engine und Audit-Bibliothek der Plattform; für Erstattungen wird kein eigener Berechtigungscode für Sonderfälle geschrieben.
Psychologische Akzeptanz	Die Freigabe wird mit einem Klick angefordert; Gründe werden aus einer kurzen Liste gewählt, damit die Kontrolle ehrliche Mitarbeitende nicht ausbremst.
Verantwortlichkeit	Jede Erstattung wird in ein Append-only-Audit-Log geschrieben, mit Mitarbeitendem, freigebender Person, Bestellung, Betrag und Grund.
Defense in Depth	Das Backoffice-Tool ist nur über Single Sign-on mit Phishing-resistenter MFA von verwalteten Geräten aus erreichbar.

Sowohl unehrliche Mitarbeitende als auch ein Angreifer, der das Konto eines Mitarbeitenden übernommen hat, sind nun auf kleine, protokollierte Erstattungen in einem Markt beschränkt.

Der Preis ist eine kurze Verzögerung bei großen Erstattungen und etwas zusätzliche Designarbeit, was weit billiger ist, als diese Kontrollen nachzurüsten, nachdem Betrug entdeckt wurde.

PRÜFEN SIE IHR VERSTÄNDNIS

▼ Warum hängt Defense in Depth davon ab, dass die Schichten unabhängig sind?

Versagen zwei Schichten aus demselben Grund, etwa weil beide demselben gefälschten Header vertrauen, überwindet ein einziger Angriff beide, sodass sie nicht mehr Schutz bieten als eine einzelne Schicht.

▼ Welches Prinzip von Saltzer und Schroeder wird verletzt, wenn ein Service prüft, ob ein Benutzer angemeldet ist, aber nicht, ob die angefragte Bestellung ihm gehört?

Vollständige Zugriffsprüfung: Jeder Zugriff auf jedes Objekt muss auf Berechtigung geprüft werden, nicht nur die Tatsache, dass der Benutzer sich authentifiziert hat.

▼ Ein Team sagt, seine Anwendung sei „OWASP-Top-10-konform“. Was ist an dieser Aussage falsch?

Die Top 10 sind eine Sensibilisierungsliste von Risikokategorien, keine Norm mit testbaren Anforderungen; für eine Überprüfung braucht man etwas wie OWASP ASVS und ein Bedrohungsmodell des tatsächlichen Systems.

▼ Nennen Sie ein Beispiel für Fail-open und erklären Sie, warum es gefährlich ist.

Eine Berechtigungsprüfung, die ein Timeout der Policy Engine als „erlaubt“ wertet. Ein Angreifer, der die Policy Engine verlangsamen oder lahmlegen kann, erhält dann Zugriff auf alles, was sie schützt.

▼ **Probieren Sie es aus: Jede Kassenkraft kann auf jede Buchung einen Rabatt von 100 % gewähren, und die App blendet die Schaltfläche für Nachwuchskräfte lediglich aus. Welche Sicherheitsprinzipien verletzt das? Entwerfen Sie es kurz neu, auch mit der Frage, wer große Rabatte freigeben darf.**

Das Ausblenden einer Schaltfläche geschieht clientseitig, also außerhalb der Vertrauensgrenze, und die vollständige Zugriffsprüfung versagt: Der Server muss jeden Rabatt prüfen. Auch Least Privilege versagt. Geben Sie dem Kassenpersonal eine begrenzte Berechtigung, etwa Rabatte bis 10 % am eigenen Veranstaltungsort, standardmäßig verweigert. Verlangen Sie für größere Rabatte eine zweite Person, was Trennung von Privilegien ist, und halten Sie fest, wer was getan hat. Arbeiten Sie Fail-closed, wenn die Berechtigungsprüfung nicht verfügbar ist, und nehmen Sie eine langsamere Warteschlange in Kauf.

QUELLEN

- OWASP Top Ten project (OWASP Top 10:2025): https://owasp.org/www-project-top-ten/
- NIST SP 800-218: Secure Software Development Framework (SSDF) version 1.1: https://csrc.nist.gov/pubs/sp/800/218/final
- Saltzer and Schroeder, The Protection of Information in Computer Systems (1975): https://doi.org/10.1109/PROC.1975.9939
- OWASP SAMM: Software Assurance Maturity Model: https://owasp.org/www-project-samm/
- OWASP DevSecOps Guideline: Dynamic Application Security Testing: https://owasp.github.io/www-project-devsecops-guideline/latest/02b-Dynamic-Application-Security-Testing
- OWASP: Fuzzing: https://community.owasp.org/Fuzzing
- Miller, Fredriksen and So: An Empirical Study of the Reliability of UNIX Utilities: https://ftp1.cs.wisc.edu/par-distr-sys/papers/fuzz.pdf
- OSS-Fuzz documentation: https://google.github.io/oss-fuzz/

Bedrohungsmodellierung: STRIDE, Angriffsbäume und Datenflussdiagramme

BEVOR SIE DAS LESEN

Dieser Abschnitt baut auf:

- 2.7 Sicherheit, Benutzbarkeit und mehr
- 4.3 Das C4-Modell
- 8.1 Sicherheitsprinzipien

Begriffe von früher, die er verwendet: Vertrauensgrenze, Containerdiagramm, Webhook, API-Gateway, Architecture Decision Record (ADR)

DIE IDEE IN KÜRZE

Bevor eine Familie in den Urlaub fährt, geht meist jemand durchs Haus und fragt, was schiefgehen könnte: Ist die Hintertür abgeschlossen, hat ein Nachbar einen Schlüssel, liegt der Ersatzschlüssel unter dem Blumentopf, stapelt sich die Post und verrät, dass niemand zu Hause ist? Dann wird für jeden Punkt entschieden, was zu tun ist: die Tür abschließen, den Nachbarn bitten, die Post hereinzuholen, den Schlüssel woanders hinlegen. Bedrohungsmodellierung ist derselbe Rundgang durch einen Softwareentwurf. Das Team zeichnet, wie Daten durch das System fließen, fragt, was ein Angreifer bei jedem Schritt tun könnte, und entscheidet, was behoben, was akzeptiert und was entfernt wird.

WAS SIE WISSEN MÜSSEN

- Bedrohungsmodellierung beantwortet die vier Fragen von Shostack: Woran arbeiten wir, was kann schiefgehen, was werden wir dagegen tun, und haben wir gute Arbeit geleistet?
- Ein Datenflussdiagramm aus externen Entitäten, Prozessen, Datenspeichern, Datenflüssen und Vertrauensgrenzen ist das übliche Modell; Datenflüsse, die Vertrauensgrenzen überqueren, verdienen die meiste Aufmerksamkeit.
- STRIDE (Kohnfelder und Garg, Microsoft, 1999) benennt sechs Bedrohungskategorien, die jeweils eine Eigenschaft verletzen: Spoofing (Authentizität), Tampering (Integrität), Repudiation (Nichtabstreitbarkeit), Information Disclosure (Vertraulichkeit), Denial of Service (Verfügbarkeit) und Elevation of Privilege (Autorisierung).
- Angriffsbäume (Schneier, 1999) modellieren das Ziel eines Angreifers mit UND- und ODER-Knoten; der billigste Pfad durch den Baum zeigt, wo eine Gegenmaßnahme am meisten zählt.

- Bewerten Sie Bedrohungen nach Wahrscheinlichkeit und Auswirkung mit schriftlicher Begründung; DREAD kam aus der Mode, weil seine Werte subjektiv waren, und CVSS bewertet bekannte Schwachstellen, keine Bedrohungen im Design.
- Jede Bedrohung endet als gemindert, beseitigt, übertragen oder ausdrücklich akzeptiert von jemandem mit der nötigen Befugnis; erstellen Sie Bedrohungsmodelle beim Entwurf und erneut, wann immer sich Vertrauensgrenzen, Daten oder Authentifizierung ändern.

Was Bedrohungsmodellierung ist

Bedrohungsmodellierung ist die Tätigkeit, einen Entwurf darauf zu untersuchen, was mit seiner Sicherheit schiefgehen könnte, und zu entscheiden, was dagegen zu tun ist, bevor Angreifer oder Incidents es für Sie herausfinden. Sie ist der wichtigste Weg, auf dem Sicherheitsprinzipien zu konkreten Entscheidungen werden, und die direkteste Antwort auf die OWASP-Kategorie Insecure Design: Mängel, die sich auch durch noch so korrektes Programmieren nicht beheben lassen, weil die nötige Kontrolle nie entworfen wurde.

Adam Shostack, der die Arbeit an der Bedrohungsmodellierung bei Microsoft leitete und das Standardwerk dazu schrieb, reduziert die Tätigkeit auf vier Fragen. **Woran arbeiten wir?** (ein Modell des Systems, meist ein Diagramm). **Was kann schiefgehen?** (Bedrohungen, systematisch gefunden). **Was werden wir dagegen tun?** (jede Bedrohung mindern, beseitigen, übertragen oder akzeptieren). **Haben wir gute Arbeit geleistet?** (das Modell, die Bedrohungen und die Nacharbeit prüfen). Jede Methode, die die vier Fragen beantwortet, ist Bedrohungsmodellierung; der Rest dieses Abschnitts beschreibt die Techniken, mit denen die meisten Teams sie beantworten.

2020 veröffentlichte eine Gruppe von Fachleuten, darunter Shostack, das Threat Modeling Manifesto, nach dem Vorbild des Agilen Manifests. Es stellt eine Kultur des Findens und Behebens von Designproblemen über Checklisten-Compliance, Menschen und Zusammenarbeit über Prozesse und Werkzeuge, einen Weg des Verstehens über eine einmalige Sicherheitsmomentaufnahme, das Durchführen von Bedrohungsmodellierung über das Reden darüber und kontinuierliche Verfeinerung über eine einmalige Lieferung. Seine Botschaft: Bedrohungsmodellierung ist eine Gewohnheit des Teams, das das System baut, kein Dokument, das Außenstehende einmal im Jahr erstellen.

Das System modellieren: Datenflussdiagramme und Vertrauensgrenzen

Die erste Frage braucht ein Modell des Systems, das zeigt, wohin Daten fließen und wo sich das Vertrauen ändert. Die klassische Wahl ist ein Datenflussdiagramm (DFD) mit fünf Arten von Elementen: **externe Entitäten** (Menschen oder Systeme außerhalb Ihrer Kontrolle, als Rechtecke gezeichnet), **Prozesse** (Code, den Sie ausführen, als Kreise oder Ellipsen

gezeichnet), **Datenspeicher** (Datenbanken, Dateien, Queues und Caches, zwischen zwei parallelen Linien gezeichnet), **Datenflüsse** (Pfeile, die zeigen, wie Daten zwischen den anderen Elementen fließen) und **Vertrauensgrenzen** (gestrichelte Linien oder Kästen, wo sich das Vertrauensniveau ändert).

DFDs eignen sich für die Bedrohungsmodellierung, weil Angriffe den Daten folgen: Jeder Datenfluss, der eine Vertrauensgrenze überquert, ist eine Stelle, an der ein Angreifer etwas einschleusen, abfangen oder wiederholt einspielen kann. Ein C4-Containerdiagramm aus Kapitel 4 ist ein guter Ausgangspunkt; ergänzen Sie die Vertrauensgrenzen, machen Sie jeden Pfeil zu einem Datenfluss mit Richtung und Beschriftung, und nehmen Sie die Speicher und Queues auf, die Containerdiagramme manchmal nur implizit lassen. Halten Sie das Diagramm auf der Ebene, auf der Bedrohungen gefunden werden: Ein Diagramm je wichtigem Feature oder Ablauf ist nützlicher als ein riesiges Diagramm von allem.

Bedrohungen mit STRIDE finden

STRIDE wurde 1999 bei Microsoft von Loren Kohnfelder und Praerit Garg als Merkhilfe entwickelt, damit Entwickler über Bedrohungen für ihre Produkte nachdenken. Jeder Buchstabe benennt eine Kategorie von Bedrohungen, und jede Kategorie ist die Verletzung einer Sicherheitseigenschaft. Das macht STRIDE nützlich: Es ist ein systematischer Denkanstoß, keine Liste bekannter Angriffe.

STRIDE und die Eigenschaft, die jede Bedrohung verletzt

Bedrohung	Verletzte Eigenschaft	Beispiel Essenslieferung	Typische Gegenmaßnahmen
Spoofing (Vortäuschen einer Identität)	Authentizität	Ein Angreifer meldet sich mit einem per Credential Stuffing erbeuteten Passwort als Kunde an; ein fremder Prozess ruft den Zahlungsservice auf und gibt sich als Bestellservice aus.	Starke Authentifizierung (MFA, Passkeys), Rate Limits, Mutual TLS und Workload Identity zwischen Services.
Tampering (Manipulation)	Integrität	Ein Client sendet eine veränderte Warenkorbsumme; jemand verändert einen Zahlungs-Webhook oder eine Datenbankzeile.	Neuberechnung auf dem Server, Signaturen und MACs, TLS, Schreibzugriff nach Least Privilege.
Repudiation (Abstreiten)	Nichtabstreitbarkeit (Verantwortlichkeit)	Ein Kunde bestreitet, eine Bestellung aufgegeben zu haben; eine Support-Kraft bestreitet, eine Erstattung ausgelöst zu haben, und nichts beweist das Gegenteil.	Audit-Logs, die an authentifizierte Identitäten gebunden sind, Zeitstempel, manipulationssichere Speicherung.
Information Disclosure (Offenlegung von Informationen)	Vertraulichkeit	Eine API gibt Kundenadressen an Kuriere heraus, die nicht an diese Kunden liefern; Fehlermeldungen verraten Interna.	Autorisierung für jedes Objekt, Datenminimierung, Verschlüsselung, allgemein gehaltene Fehlermeldungen.
Denial of Service (Dienstverweigerung)	Verfügbarkeit	Bots überfluten den Checkout während der Abendspitze; eine teure Suchanfrage erschöpft die Datenbank.	Rate Limits und Kontingente, Timeouts, Bulkheads, Autoscaling, Kostenlimits für Abfragen.
Elevation of Privilege (Rechteausweitung)	Autorisierung	Ein Kunde ruft einen Endpoint für Restaurantmanager auf; eine Injection lässt einen Angreifer Code mit den Rechten des Service ausführen.	Autorisierung mit standardmäßiger Ablehnung, Eingabevalidierung, Sandboxing, Least Privilege.

Ein praktischer Weg, STRIDE anzuwenden, ist **STRIDE per Element**: Gehen Sie jedes Element des DFD durch und stellen Sie nur die Fragen, die zu seinem Typ passen. Externe Entitäten können vorgetäuscht werden und Handlungen abstreiten; Prozesse sind allen sechs Kategorien ausgesetzt; Datenspeicher der Manipulation, der Offenlegung von Informationen und Denial of

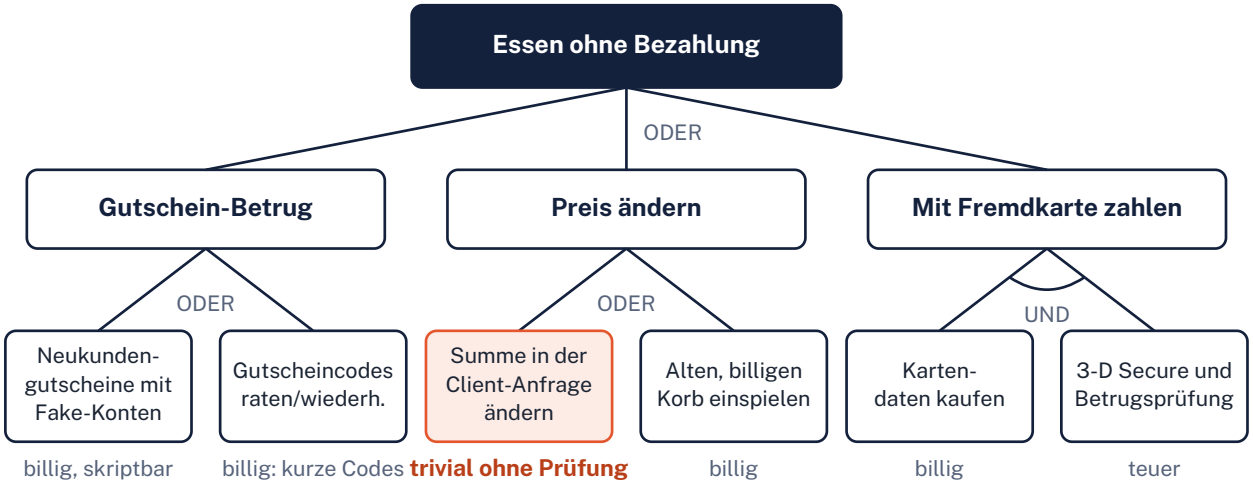
Service (und dem Abstreiten, wenn der Speicher ein Log ist); Datenflüsse der Manipulation, der Offenlegung von Informationen und Denial of Service. Wer die meiste Aufmerksamkeit auf Datenflüsse richtet, die Vertrauensgrenzen überqueren, hält die Übung fokussiert.

STRIDE findet Bedrohungen; es ordnet sie weder nach Rang noch wählt es Gegenmaßnahmen. Seine Kategorien überschneiden sich in der Praxis auch (eine vorgetäuschte Identität führt meist zur Offenlegung von Informationen oder zur Rechteausweitung), was keine Rolle spielt: Ziel ist es, die Bedrohung zu finden, nicht, sie perfekt einzuordnen. Für Datenschutzbedrohungen, etwa das Verknüpfen oder Identifizieren von Personen anhand harmlos wirkender Daten, übernimmt die Methode LINDDUN der KU Leuven die Rolle, die STRIDE für die Sicherheit spielt; sie wird bei Privacy by Design behandelt.

Angriffsbäume

Ein Angriffsbaum, bekannt gemacht durch Bruce Schneier in einem Artikel im *Dr. Dobb's Journal* von 1999, betrachtet das System aus der Sicht des Angreifers. Die Wurzel ist ein Ziel, etwa Essen zu bekommen, ohne zu bezahlen; jedes Kind ist ein Weg, seinen Elternknoten zu erreichen. Die Kinder eines ODER-Knotens sind Alternativen; alle Kinder eines UND-Knotens werden zusammen benötigt. Blätter können mit Eigenschaften wie Kosten, nötigem Können oder der Frage, ob der Schritt überhaupt möglich ist, versehen werden, und diese Werte setzen sich im Baum nach oben fort: Die Kosten eines ODER-Knotens sind die seines billigsten Kindes, die Kosten eines UND-Knotens die Summe seiner Kinder.

Angriffsbäume ergänzen STRIDE gut. STRIDE geht den Entwurf Element für Element durch und ist gut für die Abdeckung; ein Angriffsbaum folgt einem motivierten Angreifer und zeigt gut, welcher Pfad am billigsten ist und wo eine Gegenmaßnahme deshalb am meisten bringt. Im Baum unten ist das Bezahlen mit einer gestohlenen Karte relativ teuer, weil es zwei Dinge zugleich braucht, während das Ändern des Preises in der Client-Anfrage trivial ist, wenn der Server ihm vertraut. Der billigste Pfad bestimmt das Risiko, deshalb ist es mehr wert, jeden Preis auf dem Server neu zu berechnen, als noch so viel Feinabstimmung der Kartenbetrugserkennung.



Der billigste Pfad bestimmt das Risiko: Der Server muss jeden Preis neu berechnen

Ein Angriffsbaum für ein Angreiferziel; das billigste Blatt zeigt, wo eine Gegenmaßnahme am meisten zählt.

WENN DAS NEU FÜR SIE IST

Schritt für Schritt: einen Angriffsbaum lesen

Der Punkt, an dem hier die meisten stolpern, ist, wie ein Angriffsbaum zeigt, wo eine Korrektur am meisten zählt. Ein Angreifer nimmt den billigsten Weg, also ist ein Ziel nur so schwer zu erreichen wie sein billigster Pfad. Arbeiten Sie sich im Diagramm von unten nach oben durch den Baum.

- Beginnen Sie mit den Blättern, jedes ein einzelner Schritt mit groben Kosten. Gutscheine mit Fake-Konten abzugreifen ist billig; eine gestohlene Karte an 3-D Secure (der zusätzlichen Prüfung der Bank beim Bezahlen) und den Betrugsprüfungen vorbeizubringen ist teuer; die Summe in der Client-Anfrage zu ändern ist trivial, wenn der Server ihr vertraut.
- „Mit gestohlener Karte zahlen“ ist ein UND-Knoten: Der Angreifer braucht Kartendaten und einen Weg an den Prüfungen vorbei, beides zusammen. Seine Kosten sind die Summe der beiden, also macht der eine teure Schritt den ganzen Zweig teuer.
- „Preis ändern“ ist ein ODER-Knoten: Jedes seiner Blätter genügt, also sind seine Kosten die seines billigsten Kindes. Der Zweig bleibt trivial, solange der Server Preisen vertraut, die der Client sendet.
- Das Ziel an der Spitze ist ebenfalls ein ODER-Knoten und kostet so viel wie sein billigster Zweig: den Preis ändern. Das ist das eigentliche Risiko der Plattform, wie gut ihre Betrugsprüfungen auch sein mögen.
- Also lässt das Team den Server jeden Preis neu berechnen. Eine geänderte Summe bewirkt nun nichts mehr, und dieses Blatt ist keine Option mehr.
- Arbeiten Sie sich erneut nach oben. Die billigsten verbleibenden Blätter, etwa das Abgreifen von Gutscheinen mit Fake-Konten, bestimmen nun das Risiko, und dort ist die nächste Korrektur am meisten wert.
- **Die Lehre:** Einen teuren Pfad zu erschweren, ändert nichts, solange ein billiger bleibt. Finden Sie den billigsten Pfad, schließen Sie ihn und schauen Sie erneut hin, denn ein motivierter Angreifer weicht einfach auf den nächstbilligeren aus.

Risiken bewerten

Ein Bedrohungsmodell findet in der Regel mehr Bedrohungen, als ein Team auf einmal angehen kann, deshalb braucht jede eine Bewertung. Üblich ist es, **Wahrscheinlichkeit** (wie leicht, wie attraktiv, wie exponiert) und **Auswirkung** (was verloren geht, wenn es passiert: Geld, personenbezogene Daten, Verfügbarkeit, Ruf, rechtliche Risiken) auf einfachen Skalen wie niedrig, mittel und hoch einzuschätzen und in einer Matrix zu kombinieren. Der Wert der Bewertung liegt weniger in der Genauigkeit der Zahl als in einer schriftlichen Begründung, die

andere hinterfragen können. Organisationen mit reifem Risikomanagement nutzen manchmal quantitative Methoden wie FAIR (Factor Analysis of Information Risk), um Risiko als Spanne erwarteter Verluste auszudrücken.

Zwei beliebte Abkürzungen sollte man kennen, weil sie oft falsch eingesetzt werden. **DREAD** (Damage, Reproducibility, Exploitability, Affected Users, Discoverability), einst bei Microsoft im Einsatz, bewertete jede Bedrohung nach fünf Faktoren (oft von 1 bis 10) und bildete den Durchschnitt. Es kam aus der Mode, auch bei Microsoft, weil verschiedene Personen dieselbe Bedrohung sehr unterschiedlich bewerteten und die gemittelte Zahl eine falsche Genauigkeit vorspiegelte; besonders Discoverability belohnt Verschleierung, was dem offenen Design widerspricht. CVSS, das von FIRST gepflegte Common Vulnerability Scoring System (Version 4.0, 2023), ist das richtige Werkzeug für eine andere Aufgabe: den Schweregrad einer bekannten Schwachstelle in einem Produkt zu bewerten, etwa eines Fehlers in einer Bibliothek. Sein Base Score beschreibt die Schwachstelle, nicht das Risiko einer Designbedrohung in Ihrem Kontext, deshalb sollte man es nicht verwenden, um die Ergebnisse eines Bedrohungsmodells zu ordnen.

Jede Bedrohung endet in einer von vier Entscheidungen: **mindern** (eine Kontrolle hinzufügen), **beseitigen** (das Feature oder die Daten entfernen, oft die beste Option), **übertragen** (jemanden, der besser aufgestellt ist, sie tragen lassen, etwa einen Zahlungsanbieter, der die Kartendaten hält) oder **akzeptieren** (festhalten, dass eine dazu befugte Person das Restrisiko akzeptiert hat, und warum). Die häufigste Schwachstelle ist die nicht dokumentierte Akzeptanz: Die Bedrohung wurde gefunden, niemand hat sie behoben, und niemand hat entschieden, es nicht zu tun.

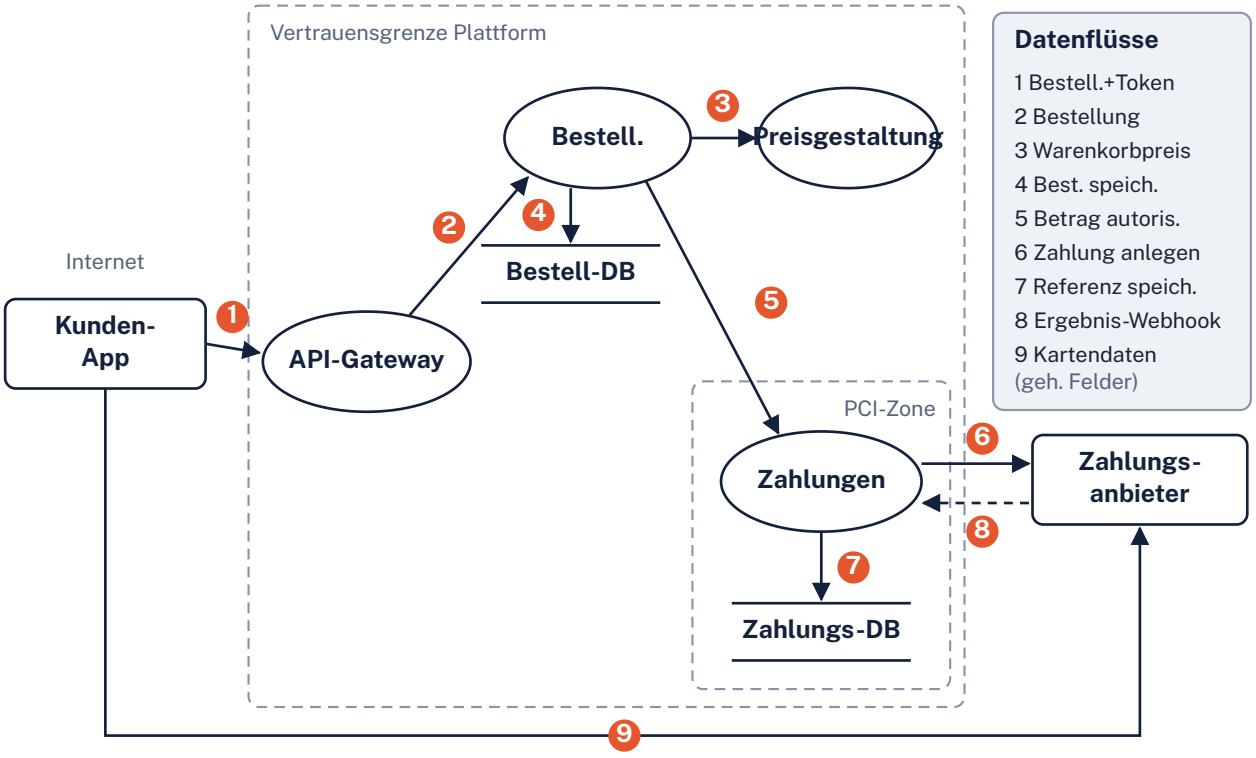
AUSGEARBEITETES BEISPIEL

Ausgearbeitetes Beispiel: ein Bedrohungsmodell für den Checkout

Das Team der Essenslieferplattform gestaltet den Bezahlvorgang neu, damit Kunden in der App mit Karte oder Wallet zahlen können. Der Architekt leitet eine 90-minütige Sitzung mit den Entwicklern von Bestellungen und Zahlungen, einem Product Owner und einem Security Champion. Das DFD unten beantwortet die erste Frage, woran wir arbeiten. Es zeigt zwei Vertrauensgrenzen: die Plattform als Ganzes und eine engere, streng abgeschottete Zahlungszone (beschriftet als PCI-Zone) um den Zahlungsservice und seine Datenbank, die Zahlungstokens und Referenzen halten und mit dem Zahlungsanbieter kommunizieren; ihre Abschottung hält den PCI-DSS-Geltungsbereich der Plattform klein. Die Kartendaten selbst (Datenfluss 9) gehen über gehostete Zahlungsfelder direkt von der App zum Zahlungsanbieter und gelangen nie in die Plattform.

1. Woran arbeiten wir?	Das Checkout-DFD unten: Kunden-App, API-Gateway, Bestellungen, Preisgestaltung, Zahlungen, zwei Datenbanken und der Zahlungsanbieter, mit zwei Vertrauensgrenzen (der Plattform und der PCI-Zone) und neun nummerierten Datenflüssen.
2. Was kann schiefgehen?	STRIDE per Element für jeden Datenfluss, der eine Grenze überquert (1, 5, 6, 8 und 9), und für die Bestell-API, dazu der weiter oben gezeigte Angriffsbaum für Essen ohne Bezahlung.
3. Was werden wir dagegen tun?	Jede Bedrohung wird bewertet und erhält eine Entscheidung (mindern, beseitigen, übertragen oder akzeptieren) und eine verantwortliche Person; die wichtigsten stehen in der Tabelle unten.
4. Haben wir gute Arbeit geleistet?	Ein Review einen Monat später vergleicht das Modell mit dem, was gebaut wurde, und prüft, ob jedes hohe Risiko gemindert oder schriftlich akzeptiert wurde.

Der billigste Weg, Kartennummern zu schützen, ist, sie nicht zu halten: Datenfluss 9 hält sie ganz aus der Plattform heraus, was Bedrohungen beseitigt, statt sie zu mindern.



Das Checkout-Datenflussdiagramm mit seinen zwei Vertrauensgrenzen und nummerierten Datenflüssen.

Ausgewählte Bedrohungen aus dem Checkout-Bedrohungsmodell

ID und Ort	STRIDE	Bedrohung	Risiko	Entscheidung und Gegenmaßnahme
T1, Datenfluss 1	S	Credential Stuffing übernimmt Konten mit gespeicherten Karten.	Hoch	Mindern: Passkeys oder MFA auf neuen Geräten; Rate Limits je Konto und IP; Prüfung auf geleakte Passwörter.
T2, Datenflüsse 1 und 2	T	Der Client sendet einen veränderten Preis, Rabatt oder eine veränderte Liefergebühr.	Hoch	Mindern: Der Bestellservice ignoriert Client-Preise; die Preisgestaltung berechnet neu aus Speisekarten und Gutscheinen; die Zahlung wird über die Summe des Servers angelegt.
T3, Datenfluss 5	S, E	Ein anderer interner Service oder ein Angreifer im Netzwerk bittet den Zahlungsservice, beliebige Beträge zu autorisieren oder zu erstatten.	Hoch	Mindern: Mutual TLS mit Workload Identity; der Zahlungsservice akzeptiert Autorisierungen nur von der Identität des Bestellservice, gebunden an eine Bestell-ID und einen Betrag.
T4, Datenfluss 8	S, T	Ein gefälschter Webhook „Zahlung erfolgreich“ markiert eine unbezahlte Bestellung als bezahlt.	Hoch	Mindern: Signatur und Zeitstempel des Anbieters prüfen; den Status über die API des Anbieters bestätigen, bevor die Bestellung freigegeben wird; jedes Event nur einmal verarbeiten.
T5, `GET /orders/{id}`	I	Das Durchprobieren von Bestell-IDs legt die Adressen anderer Kunden offen (Broken Object Level Authorization).	Hoch	Mindern: Eigentümerprüfung bei jeder Anfrage; zufällige Kennungen nur als ergänzende Maßnahme.
T6, Bestell-DB	I	Bestelldatensätze mit Adressen fließen aus einem Backup oder über ein zu weit berechtigtes Analysekonto ab.	Mittel	Mindern: Verschlüsselung im Ruhezustand mit KMS-Schlüsseln, Konten nach Least Privilege, Backups in einem separaten Konto.
T7, Datenfluss 1	D	Bots erschöpfen in der Abendspitze die Kapazität des Checkouts.	Mittel	Mindern: Rate Limits und Bot-Erkennung an der Edge, reservierte Kapazität für den Checkout, bei Bedarf ein virtueller Warteraum.

ID und Ort	STRIDE	Bedrohung	Risiko	Entscheidung und Gegenmaßnahme
T8, Bestellungen	R	Ein Kunde bestreitet, eine Bestellung aufgegeben zu haben, und es gibt keinen Nachweis.	Niedrig	Mindern: Audit-Event mit Benutzer-ID, Gerät, Zeit und Bestelldetails, aufbewahrt gemäß Policy.
T9, Datenfluss 9	I	Kartennummern, die durch die Plattform laufen, würden jede Komponente in den PCI-DSS-Geltungsbereich bringen.	Design	Beseitigen und übertragen: Gehostete Felder senden Kartendaten direkt an den Anbieter; die Plattform speichert nur Tokens und Referenzen.

Die Antworten auf die letzten beiden Fragen sind ebenso wichtig wie die Tabelle. Jede Gegenmaßnahme wird zu einem Backlog-Eintrag mit einer verantwortlichen Person; T3 und T4 werden außerdem zu Sicherheitsszenarien mit automatisierten Tests (ein Webhook mit ungültiger Signatur muss abgelehnt werden; ein Aufruf des Zahlungsservice von einer anderen Identität als der des Bestellservice muss scheitern). Die beiden Architekturentscheidungen, gehostete Zahlungsfelder und Mutual TLS zwischen Bestell- und Zahlungsservice, werden als ADRs festgehalten. Für die vierte Frage prüft das Team das Modell einen Monat später gegen das, was tatsächlich gebaut wurde, und der Security Champion kontrolliert, ob jeder Punkt mit hohem Risiko entweder gemindert oder ausdrücklich akzeptiert wurde.

Wann und wie oft man Bedrohungsmodelle erstellt

Bedrohungsmodellierung zahlt sich am meisten beim Entwurf aus, wenn eine geänderte Entscheidung nur einen Whiteboard-Schwamm kostet. Danach sollte sie wiederholt werden, wann immer sich etwas Wesentliches ändert: eine neue Vertrauensgrenze oder externe Integration, eine neue Art sensibler Daten, eine Änderung an Authentifizierung oder Autorisierung, eine neue Deployment-Umgebung oder der Wechsel von einem Monolithen zu Services. Eine sinnvolle Regel ist, „Ändert das unser Bedrohungsmodell?“ zu einer festen Frage in Design-Reviews und in der Vorlage für Architecture Decision Records zu machen und das Modell jedes Systems mindestens jährlich und nach jedem Sicherheitsvorfall zu überprüfen.

Der Aufwand sollte verhältnismäßig sein, im Sinne des risikogetriebenen Ansatzes aus Kapitel 1. Ein neuer Microservice, der Zahlungen verarbeitet, verdient eine halbtägige Sitzung mit vollständigem DFD; ein zusätzliches Feld in einer öffentlichen Speisekarte braucht fünf Minuten und einen kurzen STRIDE-Blick auf den geänderten Datenfluss. Viele Teams halten ihr

DFD als Code neben dem System (OWASP Threat Dragon und das Threat Modeling Tool von Microsoft sind kostenlose Diagrammwerkzeuge, und Bibliotheken wie OWASP pytm erzeugen Diagramme und mögliche Bedrohungen aus Code), damit das Modell zusammen mit dem System, das es beschreibt, geprüft und aktualisiert wird.

ACHTUNG

Häufige Fehler bei der Bedrohungsmodellierung

- Das ganze Unternehmen auf einmal modellieren wollen; beginnen Sie mit einem Feature, einem Ablauf oder einem Service.
- Es einmal für ein Audit tun und das Dokument abheften; das Modell muss sich mit dem System ändern.
- Es allein dem Sicherheitsteam überlassen; die Menschen, die das System bauen, wissen, wo seine Schwachstellen liegen.
- Darüber spekulieren, wer der Angreifer ist, statt darüber, was mit dem Design schiefgehen kann.
- Bedrohungen ohne Entscheidungen, Verantwortliche oder Nacharbeit auflisten oder Risiken akzeptieren, ohne festzuhalten, wer sie akzeptiert hat.
- Hunderte allgemeiner Bedrohungen aus einem Werkzeug übernehmen, ohne zu beurteilen, welche zutreffen.

VERTIEFUNG

Weitere Methoden

PASTA (Process for Attack Simulation and Threat Analysis, 2015 von Tony UcedaVélez und Marco Morana beschrieben) ist eine risikozentrierte Methode in sieben Stufen, die bei den Geschäftszielen beginnt und mit einer Angriffssimulation endet; sie passt zu Organisationen, die Bedrohungsmodellierung eng an das Geschäftsrisiko binden wollen. MITRE ATT&CK ist eine Wissensbasis realer Taktiken und Techniken von Angreifern, nützlicher für die Entwicklung von Erkennungsregeln im Sicherheitsbetrieb als für das Design. Systeme mit Machine-Learning-Modellen bringen Bedrohungen wie Prompt-Injection und das Vergiften von Trainingsdaten mit sich, die in Kapitel 12 behandelt werden.

PRÜFEN SIE IHR VERSTÄNDNIS

▼ Welche STRIDE-Kategorien gelten bei STRIDE per Element für einen Datenfluss?

Tampering, Information Disclosure und Denial of Service: Daten bei der Übertragung können verändert, gelesen oder blockiert werden.

▼ Warum ist CVSS das falsche Werkzeug, um die in einem Bedrohungsmodell gefundenen Bedrohungen zu ordnen?

CVSS bewertet den Schweregrad einer bestimmten bekannten Schwachstelle in einem Produkt; es berücksichtigt weder Ihr Design noch Ihre Assets und Ihren Kontext und misst daher nicht das Risiko einer Bedrohung auf Designebene.

▼ Wie werden in einem Angriffsbaum die Kosten eines UND-Knotens aus seinen Kindern berechnet, und warum ist das wichtig?

Sie sind die Summe der Kosten seiner Kinder, weil alle Schritte nötig sind. UND-Knoten sind deshalb meist teure Pfade, während das billigste Kind eines ODER-Knotens das Risiko bestimmt.

▼ Was ist der häufigste Fehler im vierten Schritt der Bedrohungsmodellierung?

Bedrohungen, die gefunden, aber weder behoben noch ausdrücklich akzeptiert wurden: Niemand ist für sie verantwortlich, und niemand hat entschieden, mit ihnen zu leben.

▼ Probieren Sie es aus: Der Bezahlvorgang hat einen Webhook, über den der Zahlungsanbieter den Erfolg meldet. Benennen Sie mit STRIDE die Bedrohung, wenn jeder dorthin posten kann, eine weitere Bedrohung und je eine Gegenmaßnahme. Wie endet Ihre Arbeit an jeder Bedrohung?

Spoofing: Eine gefälschte Erfolgsmeldung markiert eine unbezahlte Bestellung als bezahlt. Mindern Sie das, indem Sie die Signatur des Anbieters prüfen und die Zahlung beim Anbieter bestätigen lassen. Denial of Service: Eine Überflutung des Endpoints blockiert echte Bestätigungen; mindern Sie das mit Rate Limits und einer Queue. Tampering am Betrag wird durch die Signatur abgedeckt. Der Endpoint überquert eine Vertrauensgrenze und verdient daher Aufmerksamkeit. Jede Bedrohung muss als gemindert, beseitigt, übertragen oder von einer befugten Person ausdrücklich akzeptiert enden.

QUELLEN

- Threat Modeling Manifesto (2020): https://www.threatmodelingmanifesto.org/
- OWASP Threat Modeling Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Threat_Modeling_Cheat_Sheet.html

- Microsoft Threat Modeling Tool: STRIDE threat categories: https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats
- Bruce Schneier, Attack Trees (Dr. Dobb's Journal, December 1999): https://www.schneier.com/academic/archives/1999/12/attack_trees.html
- FIRST: Common Vulnerability Scoring System: https://www.first.org/cvss/

ABSCHNITT 8.3 VON 8.8 · FORTGESCHRITTEN · 29 MIN. LESEZEIT · ZULETZT GEPRÜFT IM NOVEMBER 2026

Identität und Zugriff: Authentifizierung, Autorisierung, OAuth 2.0 und OpenID Connect

BEVOR SIE DAS LESEN

Dieser Abschnitt baut auf:

- 0.4 Wie Computer kommunizieren
- 0.6 APIs und Integration
- 8.1 Sicherheitsprinzipien

Begriffe von früher, die er verwendet: Access Token, API-Gateway, TLS und HTTPS, HTTP

DIE IDEE IN KÜRZE

Denken Sie an ein großes Bürogebäude. Am Empfang zeigen Sie einen Lichtbildausweis und erhalten einen Besucherausweis: Das ist Authentifizierung, der Nachweis, wer Sie sind. Der Ausweis öffnet dann manche Türen und andere nicht: Das ist Autorisierung, die Entscheidung, was Sie tun dürfen. Schicken Sie einen Kurier, der für Sie ein Paket abholt, geben Sie ihm nicht Ihren eigenen Ausweis; der Empfang gibt ihm einen eingeschränkten Passierschein, der nur die Poststelle öffnet und nur heute gilt. OAuth funktioniert für Apps wie dieser eingeschränkte Passierschein. Und auch die Maschinen des Gebäudes selbst, etwa die Aufzüge und die Alarmanlage, brauchen eigene Ausweise, damit sie einander beweisen können, dass sie echt sind.

WAS SIE WISSEN MÜSSEN

- Authentifizierung stellt fest, wer aufruft, und wird am besten in einem einzigen Identity Provider zentralisiert; Autorisierung entscheidet, was jemand mit welchem Objekt tun darf, und muss dort durchgesetzt werden, wo die Daten liegen, also in jedem Service.
- NIST SP 800-63B (Revision 4, 2025) bevorzugt Länge gegenüber Kompositionsregeln, verlangt den Abgleich mit Sperrlisten gängiger und kompromittierter Passwörter, verbietet erzwungene regelmäßige Wechsel und erwartet phishing-resistente Optionen wie Passkeys.
- Serverseitige Sessions lassen sich leicht widerrufen; in sich geschlossene JWTs skalieren, können aber vor ihrem Ablauf nicht widerrufen werden, also halten Sie sie kurzlebig, legen Sie den Algorithmus fest, prüfen Sie Issuer, Audience und Ablaufzeit und halten Sie sie aus dem localStorage heraus.

- OAuth 2.0 (RFC 6749) delegiert eingeschränkten Zugriff über Tokens; Apps von Nutzern sollten den Authorization-Code-Flow mit PKCE (RFC 7636) verwenden, Services Client Credentials; OAuth 2.1 ist 2026 noch ein Internet-Draft.
- OpenID Connect ergänzt ein ID-Token, das dem Client mitteilt, wer sich angemeldet hat; das ID-Token ist für den Client bestimmt, das Access Token für die API.
- RBAC, ABAC und beziehungsbasierte Zugriffskontrolle (Google Zanzibar, 2019) passen zu unterschiedlichen Bedürfnissen; Policy-Engines wie OPA trennen Entscheidung und Durchsetzung, und Workload Identity (SPIFFE, mTLS) ersetzt gemeinsam genutzte Service-Secrets.

Authentifizierung und Autorisierung

Jede Anfrage an die Essenslieferplattform wirft zwei getrennte Fragen auf. Authentifizierung fragt *Wer ist das?* und wird durch Nachweise beantwortet: ein Passwort, einen Passkey, ein Zertifikat, ein signiertes Token. Autorisierung fragt *Darf diese Person das tun, mit diesem Objekt, jetzt?* und wird durch eine Richtlinie beantwortet: Dieser Kunde darf diese Bestellung lesen, weil er sie aufgegeben hat; dieser Kurier darf diese Abholadresse sehen, weil ihm die Lieferung zugewiesen ist. Auch HTTP hält beides auseinander: `401 Unauthorized` bedeutet, dass der Anfrage eine gültige Authentifizierung fehlt, während `403 Forbidden` bedeutet, dass der Aufrufer bekannt ist, die Aktion aber nicht darf.

Die Unterscheidung ist architektonisch wichtig, weil beides an verschiedenen Stellen gelöst wird. Authentifizierung wird am besten in einem einzigen Identity Provider (IdP) zentralisiert, dem alle Anwendungen vertrauen, sodass Passwortregeln, Multi-Faktor-Authentifizierung, Kontowiederherstellung und Sperrung nur einmal implementiert werden. Autorisierung ist von Natur aus verteilt: Nur der Bestellservice weiß, wem eine Bestellung gehört, also muss die endgültige Entscheidung über eine Bestellung dort fallen, auch wenn grobe Prüfungen schon vorher am API-Gateway stattfinden. Die meisten Verstöße gegen die Zugriffskontrolle in den OWASP Top 10 entstehen, weil beides verwechselt wird: ein Endpoint, der prüft, ob das Token gültig ist, und dann jedes Objekt zurückgibt, dessen ID er erhält.

Passwörter, Multi-Faktor-Authentifizierung und Passkeys

Passwörter sind nach wie vor das häufigste Authentifizierungsmittel, und die Digital Identity Guidelines des NIST sind die am weitesten befolgten Regeln dafür. Revision 4 von SP 800-63B, veröffentlicht im Juli 2025, kehrt mehrere lang gepflegte Gewohnheiten um. Länge zählt mehr als Zusammensetzung: Ein Passwort, das als einziger Faktor dient, muss mindestens 15 Zeichen lang sein (mindestens 8, wenn es nur als Teil einer Multi-Faktor-Authentifizierung genutzt wird), und prüfende Systeme sollten mindestens 64 Zeichen akzeptieren, einschließlich Leerzeichen und Unicode. Sie dürfen keine Kompositionsregeln vorschreiben,

etwa eine Ziffer und ein Sonderzeichen zu verlangen; solche Regeln machen Passwörter schwerer zu merken, ohne sie wesentlich schwerer zu erraten. Sie müssen neue Passwörter mit einer Sperrliste gängiger, erwartbarer und kompromittierter Passwörter abgleichen. Sie dürfen keine regelmäßigen Wechsel erzwingen, sondern nur dann einen Wechsel, wenn es Hinweise auf eine Kompromittierung gibt. Sie müssen Passwortmanager zulassen und sollten das Einfügen erlauben, dürfen keine vor der Anmeldung lesbaren Passworthinweise anbieten und nicht nach Sicherheitsfragen fragen. Fehlgeschlagene Versuche müssen per Rate Limiting begrenzt werden.

Multi-Faktor-Authentifizierung (MFA) kombiniert Faktoren unterschiedlicher Art: etwas, das Sie wissen (ein Passwort), etwas, das Sie haben (ein Smartphone oder einen Sicherheitsschlüssel), und etwas, das Sie sind (ein Fingerabdruck, meist nur zum Entsperren eines Geräts genutzt). Sie vereitelt das meiste Credential Stuffing, bei dem Angreifer Kombinationen aus Benutzername und Passwort durchprobieren, die von anderen Websites geleakt wurden. Nicht alle zweiten Faktoren sind gleichwertig: Per SMS gesendete Einmalcodes lassen sich durch SIM-Swapping stehlen, und kein Einmalcode, ob per SMS oder aus einer Authenticator-App, widersteht Phishing, weil ein Nutzer dazu gebracht werden kann, ihn auf einer gefälschten Website einzugeben, die ihn sofort weiterleitet. Die NIST-Richtlinien verlangen deshalb, dass Dienste auf Authentication Assurance Level 2 mindestens eine phishing-resistente Option anbieten und auf Level 3 eine verwenden.

Ein Passkey ist die phishing-resistente Option, die die meisten Nutzer tatsächlich verwenden können. Passkeys sind FIDO2-Anmeldedaten, aufgebaut auf der Web Authentication API (WebAuthn) des W3C und dem Client to Authenticator Protocol (CTAP) der FIDO Alliance. Bei der Registrierung erzeugt das Gerät des Nutzers ein neues Schlüsselpaar für diese Website; der Server speichert nur den öffentlichen Schlüssel. Bei der Anmeldung sendet der Server eine zufällige Challenge, und das Gerät signiert sie mit dem privaten Schlüssel, nachdem der Nutzer es per Fingerabdruck, Gesicht oder PIN entsperrt hat. Weil der Browser die Anmeldedaten an den Origin der Website bindet, kann eine täuschend ähnliche Phishing-Website keine gültige Signatur erhalten, und weil der Server kein gemeinsames Geheimnis hält, liefert ein Datenbankeinbruch nichts, was ein Angreifer wiederverwenden könnte. Passkeys können gerätegebunden sein (auf einem Hardware-Sicherheitsschlüssel) oder von einer Plattform oder einem Passwortmanager zwischen den Geräten eines Nutzers synchronisiert werden; die NIST-Leitlinien von 2025 akzeptieren synchronisierte Passkeys auf Level 2, während Level 3 einen nicht exportierbaren Schlüssel verlangt.

Die architektonische Schwachstelle jedes Authentifizierungsverfahrens ist die **Kontowiederherstellung**. Lässt sich ein verlorener Passkey durch einen Klick auf einen per E-Mail gesendeten Link ersetzen, ist das ganze System nur so stark wie das E-Mail-Konto.

Wiederherstellungsabläufe verdienen ein eigenes Bedrohungsmodell, Rate Limits, Benachrichtigungen an die anderen Geräte des Nutzers und bei wertvollen Konten Wartezeiten oder Prüfungen durch Menschen.

Sessions und Tokens

Hat sich ein Nutzer authentifiziert, muss das System seine späteren Anfragen erkennen, ohne erneut zu fragen. Es gibt zwei grundlegende Entwürfe. Bei einer **serverseitigen Session** speichert der Server die Session (Nutzer, Ablaufzeit, vielleicht Rollen) und gibt dem Browser eine lange zufällige Session-ID in einem Cookie. Bei einem Entwurf mit **in sich geschlossenen Tokens** signiert der Aussteller ein Token, das die Claims enthält, typischerweise ein JSON Web Token (JWT, RFC 7519), und jeder Service, der den öffentlichen Schlüssel des Ausstellers kennt, kann es ohne Nachschlagen validieren.

Serverseitige Sessions und in sich geschlossene Tokens im Vergleich

Aspekt	Serverseitige Session	In sich geschlossenes Token (JWT)
Wo der Zustand liegt	In einem Session-Store auf dem Server; der Browser hält nur eine zufällige ID.	Im signierten Token; der Server speichert nichts pro Nutzer.
Widerruf	Session löschen: sofort wirksam.	Vor dem Ablauf nicht möglich ohne Sperrliste oder Nachschlagen, daher müssen Laufzeiten kurz sein.
Skalierung über Services hinweg	Braucht einen gemeinsamen Session-Store oder Sticky Routing.	Jeder Service kann die Signatur lokal prüfen.
Größe bei der Übertragung	Ein kleines Cookie.	Hunderte Bytes bis Kilobytes bei jeder Anfrage.
Typische Verwendung	Browser-Sessions für Webanwendungen und Backends for Frontends.	Access Tokens für APIs, mobile Apps und Aufrufe zwischen Services.
Hauptrisiken	Session Fixation und Cross-Site Request Forgery (gemindert durch Neuerzeugen der ID bei der Anmeldung und SameSite-Cookies).	Diebstahl durch Script-Injection bei Ablage im Browser-Speicher; Fehler bei der Validierung; veraltete Claims.

Für Browseranwendungen ist eine serverseitige Session in einem Cookie mit den Attributen `HttpOnly` (für JavaScript unsichtbar), `Secure` (nur HTTPS) und `SameSite` kaum zu übertreffen. Single-Page-Anwendungen setzen zunehmend auf ein **Backend for Frontend** (BFF): Der Browser hält nur ein Session-Cookie für sein eigenes kleines Backend, das die OAuth-Tokens verwahrt und die APIs in seinem Auftrag aufruft, ein Ansatz, den die Leitlinien

der IETF-OAuth-Arbeitsgruppe für browserbasierte Anwendungen empfehlen. JWTs glänzen zwischen Services und in mobilen Apps, wo eine zustandslose Prüfung an jeder Station einen zentralen Engpass vermeidet.

ACHTUNG

Fallstricke bei JWTs

- **Kein Widerruf.** Ein gestohlenes JWT ist gültig, bis es abläuft. Halten Sie Access Tokens kurzlebig (Minuten), verwenden Sie Refresh Tokens, die der Autorisierungsserver widerrufen kann, und führen Sie für Notfälle eine Sperrliste von Token-IDs, oder verwenden Sie opake Tokens, die per Introspection (RFC 7662) geprüft werden, wo sofortiger Widerruf wichtig ist.
- **Algorithmusverwechslung.** Manche Bibliotheken akzeptierten früher den im Token-Header genannten Algorithmus, einschließlich `none`, oder prüften ein RS256-Token als HS256 mit dem öffentlichen Schlüssel als HMAC-Geheimnis, sodass Angreifer Tokens fälschen konnten. Legen Sie die erwarteten Algorithmen immer im Code fest, wie es RFC 8725 (JWT Best Current Practices) empfiehlt.
- **Tokens im `localStorage` ablegen.** Jedes Skript, das auf der Seite läuft, kann ihn lesen, sodass eine einzige Cross-Site-Scripting-Lücke das Token jedes Nutzers preisgibt. Bevorzugen Sie HttpOnly-Cookies über ein BFF, oder halten Sie Tokens im Arbeitsspeicher und binden Sie sie mit absendergebundenen Tokens wie DPoP (RFC 9449) an den Client.
- **Signiert ist nicht verschlüsselt.** Der Payload eines JWT ist lediglich Base64url-kodiert; wer das Token hat, kann ihn lesen. Legen Sie keine personenbezogenen Daten hinein.
- **Claim-Prüfungen auslassen.** Validieren Sie Issuer, Audience und Ablaufzeit, nicht nur die Signatur, sonst wird ein Token, das für eine API ausgestellt wurde, von einer anderen akzeptiert.

```python
import jwt  # PyJWT
from jwt import PyJWKClient

ISSUER = "https://id.fooddelivery.example/"
jwks = PyJWKClient(ISSUER + ".well-known/jwks.json")  # issuer's public keys

def verify_access_token(token: str) -> dict:
    signing_key = jwks.get_signing_key_from_jwt(token)
    return jwt.decode(
        token,
        signing_key.key,
        algorithms=["RS256"],      # pinned: never read from the token header
        audience="orders-api",      # the token must be meant for this API
        issuer=ISSUER,
        options={"require": ["exp", "iat", "sub"]},
    )  # raises an exception on any failure: fail closed
```

Validierung eines Access Tokens in einer Python-API mit PyJWT: Algorithmus, Audience und Issuer legt die API fest, nie das Token.

WENN DAS NEU FÜR SIE IST

Schritt für Schritt: ein Token, das niemand zurücknehmen kann

Der schwierigste Gedanke dieses Abschnitts ist, warum sich ein in sich geschlossenes Token nicht einfach stornieren lässt und was Entwerfende dagegen tun. Verfolgen Sie einen Diebstahl unter jedem der beiden Entwürfe aus der Tabelle.

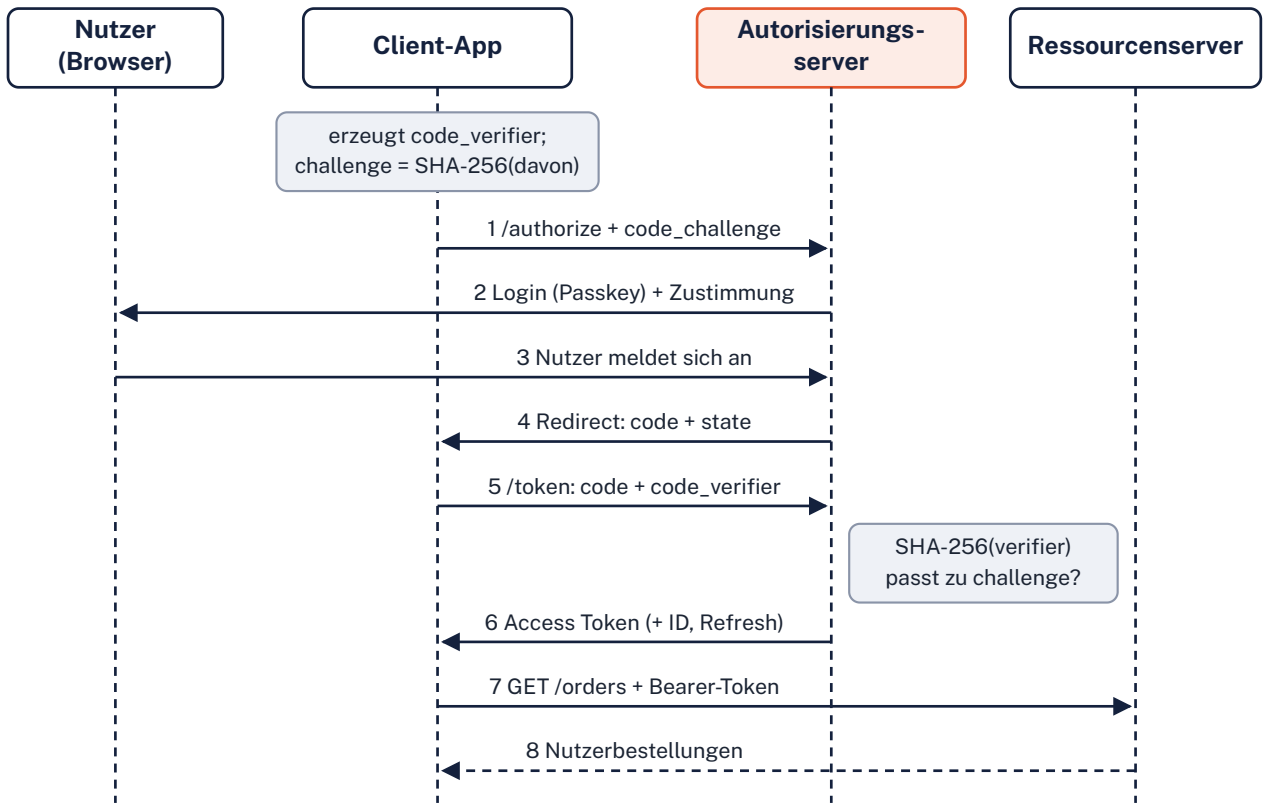
- Bei einer serverseitigen Session hält der Server die Anmeldung des Kunden in seinem Session-Store fest und gibt dem Browser nur eine lange zufällige ID in einem Cookie, die er bei jeder Anfrage nachschlägt.
- Ein Angreifer stiehlt die ID. Sobald der Diebstahl bemerkt wird, löscht die Plattform den Session-Eintrag, sodass die nächste Anfrage des Angreifers nichts findet und abgelehnt wird: Die Stornierung wirkt sofort.
- Beim anderen Entwurf hält die mobile App des Kunden ein signiertes JSON Web Token, das angibt, wer der Kunde ist und wann das Token abläuft. Der Bestellservice prüft die Signatur, ähnlich wie ein Geschäft das Wasserzeichen einer Banknote prüft, sowie Issuer, Audience und Ablaufzeit des Tokens und vertraut ihm dann, ohne jemanden zu fragen.
- Nun wird das Token gestohlen. Es gibt keinen Eintrag, den man löschen könnte, also akzeptiert jeder Service es weiterhin, weil es nach wie vor echt ist, bis es abläuft.
- Daher die Gegenmittel aus „Fallstricke bei JWTs“: Lassen Sie Access Tokens binnen Minuten ablaufen und erneuern Sie sie mit einem Refresh Token, das der Aussteller widerrufen kann; wo sofortiges Stornieren wichtig ist, führen Sie eine Sperrliste gestohlener Tokens oder verwenden Tokens, die jeder Service beim Aussteller prüfen muss, was genau das Nachschlagen zurückbringt, das Tokens vermeiden.
- Es ist also eine Abwägung: Tokens lassen jeden Service einen Aufrufer selbstständig prüfen, was zu mobilen Apps und Aufrufen zwischen Services passt; Sessions lassen sich leicht stornieren und passen zu Browseranwendungen.
- **Die Lehre:** Für sich allein ist ein in sich geschlossenes Token ein Versprechen, das vor seinem Ablauf nicht zurückgezogen werden kann. Halten Sie das Versprechen kurz und ergänzen Sie ein Nachschlagen nur dort, wo sofortiges Stornieren wichtig ist.

OAuth 2.0: delegierte Autorisierung

OAuth 2.0 (RFC 6749, 2012) löst ein bestimmtes Problem: einer Anwendung zu erlauben, im Namen eines Nutzers zu handeln, ohne dessen Passwort zu erfahren. Davor fragte ein Dienst, der Ihre Kontakte importieren wollte, nach Ihrem E-Mail-Passwort. OAuth definiert vier Rollen: den **Ressourcenbesitzer** (den Nutzer), den **Client** (die Anwendung, die Zugriff möchte), den **Autorisierungsserver** (der den Nutzer authentifiziert, um Zustimmung bittet und Tokens ausstellt) und den **Ressourcenserver** (die API mit den Daten). Der Client erhält ein **Access**

Token, das durch **Scopes** (etwa `orders:read`), eine Laufzeit und eine Audience begrenzt ist, und oft ein **Refresh Token**, um neue Access Tokens zu erhalten, ohne den Nutzer erneut zu fragen.

OAuth definiert mehrere Wege, Grants genannt, um ein Token zu erhalten. Für jede Anwendung, die für einen Nutzer handelt, ob serverseitige Web-App, Single-Page-App oder mobile App, ist der empfohlene Grant der **Authorization-Code-Flow mit PKCE** (Proof Key for Code Exchange, RFC 7636, 2015). Der Client erzeugt zuerst einen zufälligen `code_verifier` und sendet nur dessen SHA-256-Hash, die `code_challenge`, wenn er den Nutzer zum Autorisierungsserver umleitet. Nachdem sich der Nutzer angemeldet und zugestimmt hat, leitet der Autorisierungsserver mit einem kurzlebigen Autorisierungscode zurück. Der Client tauscht den Code dann in einer direkten Back-Channel-Anfrage gegen Tokens ein und sendet dabei den ursprünglichen Verifier; der Server prüft, ob dessen Hash zur Challenge passt. Ein Angreifer, der den Code abfängt, etwa eine bösartige mobile App, die für dieselbe Redirect-Adresse registriert ist, kann ihn ohne den Verifier nicht einlösen, und der hat den Client nie verlassen.



Der Authorization-Code-Flow mit PKCE: Nur der Hash des Verifiers wird in den Browser-Redirects (dem Front Channel) übertragen; der Verifier selbst geht direkt an den Token-Endpoint.

Der **Client-Credentials-Grant** deckt den Fall ohne Nutzer ab: Ein Service erhält ein Token in eigener Befugnis, zum Beispiel der Dispositionsservice, der die API eines Kartenpartners aufruft, oder das System eines Partners, das die Bestellstatus-API der Plattform aufruft. Der Client sollte sich am Token-Endpoint mit einem privaten Schlüssel (einer signierten JWT-Assertion) oder Mutual TLS (RFC 8705) authentifizieren statt mit einem gemeinsamen Geheimnis. Von zwei älteren Grants wird inzwischen abgeraten: dem **Implicit Grant**, der Tokens in der URL des Browsers zurückgab, und dem **Resource Owner Password Credentials Grant**, bei dem der Client das Passwort des Nutzers einsammelte und damit den Zweck von OAuth zunichtemachte. RFC 9700, die im Januar 2025 veröffentlichte OAuth 2.0 Security Best Current Practice, hält diese Lehren fest, darunter den exakten Abgleich von Redirect-Adressen und absendergebundene Tokens.

OAuth 2.1 fasst OAuth 2.0 und diese bewährten Praktiken in einem Dokument zusammen: PKCE ist für den Authorization-Code-Flow Pflicht, Implicit und Password Grant entfallen, Redirect-Adressen müssen exakt übereinstimmen, und Bearer-Tokens dürfen nicht mehr in URL-Query-Strings gesendet werden. Bei der letzten Prüfung dieses Abschnitts war es noch ein IETF-Internet-Draft, kein RFC (prüfen Sie den aktuellen Status), doch seine Regeln beschreiben bereits, was gut betriebene Autorisierungsserver tun.

ENTSCHEIDUNGSHILFE

Entscheidungshilfe: welcher OAuth-2.0-Flow?

Für jede Art von Aufrufer der APIs der Essenslieferplattform.

Q1 **Kann das Gerät keinen Browser anzeigen oder ist es schwer zu bedienen, etwa das Küchendisplay eines Restaurants?**

- **Ja** Verwenden Sie den Device Authorization Grant (RFC 8628): Das Gerät zeigt einen kurzen Code an, den der Nutzer auf seinem Smartphone bestätigt.
- **Nein** Weiter zur nächsten Frage.

Q2 **Ist eine Person anwesend, und handelt die Anwendung in ihrem Namen?**

- **Ja** Verwenden Sie den Authorization-Code-Flow mit PKCE, gleichermaßen für Web-, Single-Page- und mobile Apps; nie den Implicit oder den Password Grant.
- **Nein** Weiter zur nächsten Frage.

Q3 **Ruft einer Ihrer eigenen Services einen anderen innerhalb Ihrer Plattform auf?**

- **Ja** Bevorzugen Sie Workload Identity (Mutual TLS mit von der Plattform ausgestellten Identitäten); erwartet die API OAuth-Tokens, verwenden Sie Client Credentials mit schlüsselbasierter Client-Authentifizierung.
- **Nein** Weiter zur nächsten Frage.

Q4 **Ruft das System eines Partners Ihre API in eigener Befugnis auf?**

- **Ja** Verwenden Sie Client Credentials mit eng begrenzten Scopes und kurzlebigen Tokens, einem Client pro Partner und Rate Limits pro Partner.
- **Nein** Überprüfen Sie die Anforderung: Ein Entwurf, in dem Ihr Client das Passwort eines Nutzers braucht, ist ein Warnsignal.

OpenID Connect, Single Sign-on und SAML

OAuth stellt Access Tokens für APIs aus; es sagt dem Client für sich allein nicht, wer der Nutzer ist. Ein Access Token als Anmeldenachweis zu verwenden, ist ein bekannter Fehler, weil ein Token, das für eine Anwendung ausgestellt wurde, bei einer anderen wiederverwendet werden kann, die seine Audience nicht prüft. OpenID Connect (OIDC, Core 1.0, 2014) ergänzt die fehlende Identitätsschicht. Mit dem Scope `openid` gibt der Autorisierungsserver, der nun als OpenID Provider auftritt, zusätzlich ein **ID-Token** zurück: ein signiertes JWT für den Client, das den Issuer angibt, die stabile Kennung des Nutzers (`sub`), die Audience (die Client-ID), Ausstellungs- und Ablaufzeit, eine `nonce` , die es an diese Anmeldung bindet, und optional, wie und wann sich der Nutzer authentifiziert hat. OIDC standardisiert außerdem einen Userinfo-

Endpoint und Discovery-Dokumente, die die Endpoints und Signaturschlüssel des Providers veröffentlichen. Die Faustregel: Das ID-Token ist für den Client, damit er weiß, wer sich angemeldet hat; das Access Token ist für die API, um Aufrufe zu autorisieren.

Single Sign-on (SSO) ergibt sich daraus fast von selbst: Hat ein Nutzer eine Session beim Identity Provider, kann jede Anwendung, die ihm vertraut, ihn ohne erneute Passwortabfrage anmelden, und die Deaktivierung des Kontos an einer Stelle sperrt den Zugriff überall. In Unternehmen nutzt SSO oft noch **SAML 2.0** (ein OASIS-Standard von 2005), der signierte XML-Assertions über den Browser austauscht. Die Essenslieferplattform könnte OIDC für ihre eigenen Apps verwenden und SAML von großen Restaurantketten akzeptieren, deren Personal sich mit seinen Unternehmenskonten im Restaurantportal anmelden soll. Das Anlegen und Entfernen von Konten über Systeme hinweg wird meist mit SCIM (RFC 7643 und 7644) automatisiert.

Autorisierungsmodelle: RBAC, ABAC und Beziehungen

Authentifizierung sagt dem System, wer aufruft; ein Autorisierungsmodell entscheidet, was der Aufrufer tun darf. Drei Modelle dominieren, und reale Systeme kombinieren sie meist.

Drei Autorisierungsmodelle

Modell	Wie es entscheidet	Stärken	Schwächen
Rollenbasiert (RBAC)	Berechtigungen werden Rollen zugewiesen; Nutzern werden Rollen zugewiesen. 1992 von David Ferraiolo und Richard Kuhn am NIST formalisiert.	Leicht zu verstehen und zu prüfen; bildet Tätigkeiten wie Supportmitarbeitende ab.	Kann „nur die eigenen“ ohne Zusatzregeln nicht ausdrücken; Eingrenzung nach Filiale oder Markt führt zur Rollenexplosion.
Attributbasiert (ABAC)	Eine Richtlinie wertet Attribute von Subjekt, Ressource, Aktion und Umgebung aus; beschrieben in NIST SP 800-162 (2014).	Sehr ausdrucksstark: Markt, Datenklassifizierung, Uhrzeit, Gerätezustand, Betrag.	„Wer kann darauf zugreifen?“ ist schwerer zu beantworten; Richtlinien können undurchsichtig und langsam werden.
Beziehungsbasiert (ReBAC)	Der Zugriff folgt Beziehungen, die als Graph gespeichert sind: Ein Nutzer leitet eine Filiale, eine Filiale gehört zu einem Restaurant, eine Bestellung gehört zu einer Filiale.	Natürlich für Freigaben und Hierarchien; skaliert bis zu feingranularen Berechtigungen pro Objekt.	Braucht einen eigenen, konsistenten Beziehungsspeicher und sorgfältige Modellierung.

Beziehungsbasierte Zugriffskontrolle ist vor allem durch Googles Paper von 2019 über **Zanzibar** bekannt, das globale Autorisierungssystem hinter Google Drive, Calendar, YouTube und vielen anderen Produkten. Zanzibar speichert Relationstupel der Form *object#relation@user*, wertet Berechtigungen aus, indem es ihnen folgt, und verwendet Konsistenz-Tokens, damit eine Berechtigungsprüfung nie gegen Daten ausgewertet wird, die älter sind als der Inhalt, den sie schützt; das ist das sogenannte New-Enemy-Problem: Wer jemanden aus einem Dokument entfernt und dann vertrauliche Inhalte hinzufügt, darf ihn diese nicht sehen lassen. Davon inspirierte Open-Source-Systeme sind unter anderem SpiceDB, OpenFGA und Ory Keto.

Unabhängig vom Modell hilft es, den Policy Decision Point (PDP), der die Richtlinie auswertet und mit „erlauben“ oder „verweigern“ antwortet, vom **Policy Enforcement Point** (PEP) zu trennen, dem Code im Gateway oder Service, der die Frage stellt und die Antwort durchsetzt. Universelle Policy-Engines wie Open Policy Agent (OPA, ein Graduated-Projekt der Cloud Native Computing Foundation, mit Richtlinien in der Sprache Rego) und Cedar (2023 von Amazon Web Services als Open Source veröffentlicht) lassen Richtlinien in versionierten Dateien leben, wie Code testen und im Prozess oder als Sidecar nahe am Service auswerten. Eine sinnvolle Schichtung auf der Plattform ist: Das API-Gateway weist Anfragen ohne gültiges Token oder ohne passenden Scope ab (grob), und jeder Service entscheidet Fragen auf Objektebene mit Daten, die nur er hat (fein). Broken Object Level Authorization ist genau deshalb das erste Risiko in den OWASP API Security Top 10 (Ausgabe 2023), weil die feine Prüfung diejenige ist, die Teams vergessen.

AUSGEARBEITETES BEISPIEL

Ausgearbeitetes Beispiel: wer eine Speisekarte ändern, eine Bestellung als fertig markieren oder eine Abholadresse sehen darf

Die Restaurants auf der Plattform reichen vom einzelnen Café bis zu Ketten mit Dutzenden Filialen. Anforderungen: Kettenbesitzer verwalten jede Filiale; Filialleiter bearbeiten die Speisekarte ihrer eigenen Filiale; Küchenpersonal markiert die Bestellungen seiner Filiale als fertig; ein Kurier sieht die Abholdaten nur für Bestellungen, die ihm zugewiesen sind; Filialleiter dürfen Kulanzerstattungen bis £20 gewähren.

Erster Versuch: RBAC	Rollen `owner`, `manager`, `staff`, `courier`. Jede Prüfung braucht dann „für welche Filiale?“ oder „welche Bestellung?“, was zu Rollen wie `manager_branch_1742` führt: Tausende Rollen, eine pro Filiale und Tätigkeit.
ReBAC-Tupel	`restaurant:bella#owner@user:ana`; `branch:bella-soho#parent@restaurant:bella`; `branch:bella-soho#manager@user:ben`; `branch:bella-soho#staff@user:kai`; `order:9812#branch@branch:bella-soho`; `order:9812#courier@user:cara`
Modellregeln	`branch.edit_menu` = manager oder owner des übergeordneten Restaurants. `order.mark_ready` = staff oder manager der Filiale der Bestellung. `order.view_pickup` = der zugewiesene Kurier oder staff der Filiale der Bestellung.
Prüfungen	Darf ben die Speisekarte von bella-soho bearbeiten? Ja (manager). Darf ana? Ja (owner des übergeordneten Restaurants). Darf cara die Abholdaten von Bestellung 9812 sehen? Ja, bis die Disposition ihr Kuriertupel löscht, wenn die Lieferung abgeschlossen ist; danach nicht mehr.
ABAC für das Erstattungslimit	Die Erstattungsregel hängt von einem Attribut der Anfrage ab, nicht von einer Beziehung: Die Policy-Engine erlaubt `refund`, wenn der Nutzer die Filiale der Bestellung leitet und der Betrag höchstens £20 beträgt.
Wo durchgesetzt	Das Gateway prüft Gültigkeit und Scope des Tokens; Speisekarten-, Bestell- und Dispositionsservice fragen jeweils vor dem Handeln den PDP nach der Entscheidung auf Objektebene.

Beziehungen beantworten Audit-Fragen direkt: Alle aufzulisten, die die Speisekarte einer Filiale bearbeiten dürfen, ist eine Graphabfrage, kein Code-Review.

Auch Beziehungsdaten haben nun einen Eigentümer: Die Disposition schreibt Kuriertupel, das Restaurantportal schreibt Personaltupel, und veraltete Tupel sind ein Sicherheitsfehler.

Workload Identity: wenn der Aufrufer ein Service ist

Services authentifizieren sich weit öfter gegenüber anderen Services, als Menschen sich anmelden, und doch sind Service-Zugangsdaten oft die schwächsten: ein langlebiger API-Schlüssel in einer Umgebungsvariable, von mehreren Services geteilt und nie rotiert. Gelangt dieser Schlüssel über ein Log, ein Repository oder ein Container-Image nach außen, wird der Angreifer zum Service. Workload Identity ersetzt solche Secrets durch Identitäten, die die Plattform dem laufenden Workload ausstellt, nachdem sie geprüft hat, was der Workload ist.

Der offene Standard ist **SPIFFE** (Secure Production Identity Framework for Everyone), ein Graduated-Projekt der Cloud Native Computing Foundation. Er benennt jeden Workload mit einem URI wie `spiffe://fooddelivery.example/ns/prod/sa/orders` und stellt ein kurzlebiges, automatisch rotiertes Dokument als Nachweis aus, eine sogenannte SVID, meist ein X.509-Zertifikat (oder ein JWT). Die Referenzimplementierung SPIRE attestiert Knoten und Workload (zum Beispiel dessen Kubernetes-Namespace und Service-Account), bevor sie die Identität ausstellt. Services nutzen die Zertifikate dann für Mutual TLS, sodass der Zahlungsservice weiß, dass der Aufrufer wirklich der Bestellservice ist, und anhand der Identität autorisieren kann. Cloud-Plattformen bieten Entsprechungen (Rollen für Workloads, Managed Identities, Workload Identity Federation), und CI-Systeme wie GitHub Actions können ein signiertes OIDC-Token gegen kurzlebige Cloud-Zugangsdaten eintauschen, sodass Pipelines keine gespeicherten Schlüssel brauchen. Wie man mit diesen Identitäten ein Netzwerk ohne implizites Vertrauen aufbaut, ist Thema von Zero Trust.

Just-in-Time- und Break-Glass-Administratorzugriff

Ein dauerhaftes Administratorkonto ist ein dauerhaftes Ziel. Beim **Just-in-Time**-Zugriff hat ein Engineer keine ständigen Rechte, nur eine Berechtigung auf Abruf: Um eine Produktionsdatenbank zu reparieren, aktiviert er eine Rolle für begrenzte Zeit, nennt einen Grund, besteht die Multi-Faktor-Authentifizierung und holt bei sensiblen Rollen eine Genehmigung ein, und jeder Schritt wird protokolliert. Microsoft Entra Privileged Identity Management ist ein Beispiel. **Break-Glass**-Konten decken den Tag ab, an dem dieser Prozess versagt, etwa weil der Identity Provider ausgefallen ist. Microsofts Leitlinie lautet: mindestens zwei Konten, getrennt von den Abhängigkeiten der normalen Anmeldung, geschützt durch Phishing-resistente Authentifizierung, mit einem Alarm bei jeder Nutzung und regelmäßig getestet. Jede Nutzung wird anschließend geprüft, und der Notfallweg ist selbst ein zu steuerndes Risiko.

Autorisierung auf Message-Topics

Authentifizierung am Broker genügt nicht: Ereignisse (Abschnitt 6.4) lösen reale Aktionen aus, ein Service, der ein gefälschtes `PaymentCaptured`-Ereignis veröffentlichen kann, kann also dafür sorgen, dass unbezahlte Bestellungen ausgeliefert werden. Vergeben Sie Berechtigungen pro Topic und pro Richtung: Nur Zahlungen darf in dieses Topic schreiben, und nur benannte Consumer dürfen es lesen, was auch begrenzt, wer darin enthaltene personenbezogene Daten sieht. Apache Kafka drückt das als Access Control Lists aus, die festlegen, dass ein Principal eine Operation wie Read oder Write auf einer Ressource ausführen darf; mit einem konfigurierten Authorizer steht eine Ressource ohne passende Liste nur Super Usern zur Verfügung, sofern `allow.everyone.if.no.acl.found` nicht gesetzt ist. Standardwerte unterscheiden sich: In RabbitMQ sind Topic-Berechtigungen optional. Prüfen Sie die Dokumentation Ihres Brokers und halten Sie diese Regeln in der Versionskontrolle.

PRÜFEN SIE IHR VERSTÄNDNIS

▼ Eine API gibt einem Aufrufer mit gültigem Token 403 zurück. Was sagt Ihnen das?

Der Aufrufer wurde authentifiziert, ist aber für diese Aktion oder dieses Objekt nicht autorisiert; 401 würde fehlende oder ungültige Authentifizierung bedeuten.

▼ Wovor schützt PKCE im Authorization-Code-Flow?

Vor dem Abfangen oder Einschleusen des Autorisierungscodes: Ohne den Code-Verifier, der nie in den Browser-Redirects übertragen wird, lässt sich ein gestohlener Code nicht gegen Tokens eintauschen.

▼ Warum sollte ein Service ein ID-Token von OpenID Connect nicht als Access Token für eine API akzeptieren?

Das ID-Token ist an den Client gerichtet (seine Audience ist die Client-ID), um ihm mitzuteilen, wer sich angemeldet hat; APIs sollten nur Access Tokens akzeptieren, die für sie ausgestellt wurden, geprüft auf Audience und Scope.

▼ Warum deutet eine Rolle wie manager_branch_1742 auf das falsche Autorisierungsmodell hin?

Sie kodiert eine Beziehung (Leiter einer bestimmten Filiale) als Rolle, was zur Rollenexplosion führt; beziehungs- oder attributbasierte Regeln drücken sie direkt aus.

▼ **Probieren Sie es aus: Wählen Sie den OAuth-2.0-Flow für: die native mobile App eines Restaurants, die vom Personal genutzt wird, und einen nächtlichen Backend-Job, der die Zahlungs-API aufruft. Eine Entwicklerin sendet außerdem das ID-Token an die Bestell-API. Geben Sie zu allen drei Punkten Rat.**

Die mobile App handelt für einen Nutzer und kann kein Geheimnis wahren, verwenden Sie also den Authorization Code Flow mit PKCE. Der nächtliche Job hat keinen Nutzer, verwenden Sie also Client Credentials oder, besser, Workload Identity. Der Entwickler irrt: Das ID-Token sagt dem Client, wer sich angemeldet hat, und die API sollte das Access-Token erhalten und Aussteller, Audience und Ablauf prüfen. Halten Sie Tokens kurzlebig, und lassen Sie jeden Service weiterhin seine eigenen Prüfungen auf Objektebene durchführen.

QUELLEN

- RFC 6749: The OAuth 2.0 Authorization Framework: https://www.rfc-editor.org/rfc/rfc6749
- RFC 7636: Proof Key for Code Exchange by OAuth Public Clients: https://www.rfc-editor.org/rfc/rfc7636
- RFC 9700: Best Current Practice for OAuth 2.0 Security: https://www.rfc-editor.org/rfc/rfc9700
- NIST SP 800-63B-4: Authentication and Authenticator Management: https://pages.nist.gov/800-63-4/sp800-63b.html
- Zanzibar: Google's Consistent, Global Authorization System (USENIX ATC 2019): https://research.google/pubs/zanzibar-googles-consistent-global-authorization-system/
- Microsoft Entra Privileged Identity Management: https://learn.microsoft.com/entra/id-governance/privileged-identity-management/pim-configure
- Microsoft: Manage emergency access accounts: https://learn.microsoft.com/en-us/entra/identity/role-based-access-control/security-emergency-access
- Apache Kafka: Authorization and ACLs: https://kafka.apache.org/44/security/authorization-and-acls/
- RabbitMQ: Authentication, authorisation and access control: https://www.rabbitmq.com/docs/access-control

Verschlüsselung, Schlüsselverwaltung und Secrets

BEVOR SIE DAS LESEN

Dieser Abschnitt baut auf:

- 0.4 Wie Computer kommunizieren
- 0.5 Daten speichern
- 8.3 Identität und Zugriff

Begriffe von früher, die er verwendet: TLS und HTTPS, Workload Identity, Mutual TLS (mTLS), Passkey, JSON Web Token (JWT)

DIE IDEE IN KÜRZE

Denken Sie an ein Hotel. Die Wertsachen der Gäste kommen in Zimmersafes, und die Safes sind nur so gut wie die Kontrolle darüber, wer die Codes kennt. Das Hotel führt nicht die Codes aller Gäste in einem Buch an der Rezeption; der Generalschlüssel liegt in einem verschlossenen Schrank, den nur die Hotelleitung öffnen kann, und jedes Öffnen wird in einem Protokoll vermerkt. Die wertvollsten Stücke verwahrt das Hotel gar nicht selbst: Es gibt dem Gast eine nummerierte Quittung, und die Bank um die Ecke verwahrt das Stück. Datenschutz funktioniert genauso: Verschlüsseln Sie die Daten, bewachen Sie die Schlüssel sorgfältiger als die Daten, und halten Sie die sensibelsten Daten nach Möglichkeit gar nicht erst.

WAS SIE WISSEN MÜSSEN

- Jede Verschlüsselungsschicht schützt vor bestimmten Parteien: TLS vor dem Netzwerk, Speicherverschlüsselung vor gestohlenen Datenträgern und Backups, Verschlüsselung auf Feldebene vor Datenbankadministratoren und nachgelagerten Kopien; Schutz läuft immer auf die Kontrolle über die Schlüssel hinaus.
- Symmetrische Algorithmen (AES-GCM) verschlüsseln große Datenmengen; asymmetrische Kryptografie liefert Schlüsselvereinbarung und digitale Signaturen; die ersten Post-Quanten-Standards des NIST (FIPS 203 bis 205, 2024) machen Krypto-Agilität zu einem architektonischen Anliegen.
- Hashing, MACs, Signaturen, Verschlüsselung und Kodierung sind verschiedene Werkzeuge; Passwörter brauchen langsames, gesalzenes Passwort-Hashing wie Argon2id mit den aktuellen Parametern von OWASP oder, besser noch, Passkeys über einen Identity Provider.

- Envelope Encryption verpackt Datenschlüssel pro Datensatz unter einem Key-Encryption-Key, der in einem KMS auf Basis von HSMs liegt; das macht Rotation günstig, begrenzt den Blast Radius und ermöglicht Crypto-Shredding, um den Preis einer kritischen Abhängigkeit.
- Secrets gehören in einen Vault, vorzugsweise als kurzlebige dynamische Zugangsdaten oder ersetzt durch Workload Identity; ein geleaktes Secret muss zuerst widerrufen und danach bereinigt werden.
- Tokenisierung über einen Zahlungsanbieter hält Kartennummern von der Plattform fern und verkleinert den Geltungsbereich von PCI DSS v4.0.1; ein Datenklassifizierungsschema entscheidet, wo sich Verschlüsselung auf Feldebene trotz ihrer Kosten lohnt.

Was Verschlüsselung schützt und was nicht

Verschlüsselung wird oft als Kästchen zum Abhaken behandelt: „Die Daten werden bei der Übertragung und im Ruhezustand verschlüsselt.“ Die Formulierung verdeckt die eigentlich wichtige Frage, nämlich **gegen wen**. Jede Verschlüsselungsschicht schützt Daten vor manchen Parteien und vor anderen nicht, und in jedem Fall ist der Schutz genau so stark wie die Kontrolle darüber, wer den Schlüssel verwenden kann. Verschlüsselung beseitigt das Problem der Zugriffskontrolle nicht; sie verlagert es von den Daten zu den Schlüsseln, wo es sich meist leichter verwalten, prüfen und widerrufen lässt.

Verschlüsselung **bei der Übertragung** schützt Daten, die sich durch ein Netzwerk bewegen, davor, unterwegs gelesen oder verändert zu werden. Im Web bedeutet das TLS, eingeführt in Kapitel 0; zwischen Services bedeutet es TLS oder Mutual TLS, behandelt bei Zero Trust. Verschlüsselung **im Ruhezustand** schützt gespeicherte Daten, doch „im Ruhezustand“ ist nicht eine einzige Sache: Die Tabelle zeigt die üblichen Schichten auf der Essenslieferplattform.

Verschlüsselungsschichten und die Bedrohungen, gegen die jede wirkt

Schicht	Schützt vor	Schützt nicht vor
TLS bei der Übertragung	Abhören und Manipulation im Netzwerk, auch innerhalb des Rechenzentrums.	Einem kompromittierten Service oder Endpoint, der die Daten im Klartext sieht.
Festplatten- oder Volume-Verschlüsselung (Standard bei den großen Clouds)	Gestohlenen oder entsorgten Festplatten, an den Anbieter zurückgegebener Hardware.	Jedem, der über das laufende Betriebssystem oder die Datenbank an die Daten gelangt.
Verschlüsselung durch Datenbank oder Speicherdienst (zum Beispiel Transparent Data Encryption)	Kopien von Datendateien, Snapshots und Backups auf Speicherebene.	Jedem Datenbankkonto, das Abfragen ausführen kann, einschließlich des Kontos der Anwendung selbst und eines Angreifers per SQL-Injection.
Verschlüsselung auf Anwendungsebene (Feldebene)	Datenbankadministratoren, zu weit gefassten Konten, Analysekopien und geleakten Backups, weil sie nur Chiffretext halten.	Der Anwendung selbst, falls kompromittiert, da sie den Schlüssel verwenden kann; die Datenbank kann das Feld nicht mehr durchsuchen oder sortieren.
Ende-zu-Ende-Verschlüsselung	Jedem Vermittler, auch dem Betreiber der Plattform.	Nichts dazwischen, doch dann kann die Plattform mit den Daten nichts Nützliches anfangen.

Liest man die Tabelle von oben nach unten, schützt jede Schicht vor mehr Parteien und kostet mehr Funktionalität. Ein solider Entwurf setzt die günstigen, transparenten Schichten überall ein (TLS und Speicherverschlüsselung sind heute fast kostenlos) und die teuren gezielt, für die Daten, deren Offenlegung den größten Schaden anrichten würde. Welche Daten das sind, entscheidet die Datenklassifizierung, die später in diesem Abschnitt zusammen mit Tokenisierung und einem ausgearbeiteten Beispiel behandelt wird.

Symmetrische und asymmetrische Kryptografie

Symmetrische Verschlüsselung verwendet einen einzigen geheimen Schlüssel zum Ver- und Entschlüsseln. Sie ist schnell genug für große Datenmengen, und der Standardalgorithmus ist AES (FIPS 197, 2001) mit Schlüsseln von 128 oder 256 Bit. Moderne Entwürfe verwenden **authentifizierte Verschlüsselung**, die neben dem Verbergen des Inhalts auch Manipulationen erkennt: AES im Galois/Counter Mode (AES-GCM) oder ChaCha20-Poly1305 (RFC 8439). Authentifizierte Modi brauchen für jede unter demselben Schlüssel verschlüsselte Nachricht eine eindeutige Nonce; die Wiederverwendung einer Nonce bei AES-GCM kann Klartext offenlegen und Fälschungen ermöglichen, einer von vielen Gründen, gut getestete Bibliotheken zu verwenden, statt Primitive selbst zusammenzusetzen.

Asymmetrische Kryptografie verwendet ein Schlüsselpaar: einen öffentlichen Schlüssel, den jeder haben darf, und einen geheim gehaltenen privaten Schlüssel. Sie ermöglicht zwei Dinge, die symmetrische Kryptografie nicht kann. **Schlüsselvereinbarung** lässt zwei Parteien, die sich nie begegnet sind, über ein offenes Netzwerk einen gemeinsamen symmetrischen Schlüssel festlegen (Elliptic-Curve-Diffie-Hellman in TLS 1.3). **Digitale Signaturen** (RSA, ECDSA, Ed25519) lassen jeden mit dem öffentlichen Schlüssel prüfen, dass Daten vom Inhaber des privaten Schlüssels stammen und nicht verändert wurden; das liefert Integrität, Authentizität und eine Grundlage für Nichtabstreitbarkeit. Asymmetrische Operationen sind langsam, daher sind reale Protokolle hybrid: TLS nutzt asymmetrische Kryptografie, um den Server zu authentifizieren und Schlüssel zu vereinbaren, und dann symmetrische Verschlüsselung für die Daten.

Ein großer Quantencomputer würde die heutigen RSA- und Elliptic-Curve-Algorithmen brechen, nicht aber AES mit ausreichend langen Schlüsseln. Im August 2024 veröffentlichte das NIST seine ersten Post-Quanten-Standards: FIPS 203 (ML-KEM, für den Schlüsselaufbau), FIPS 204 (ML-DSA) und FIPS 205 (SLH-DSA), beide für Signaturen. Daten, die viele Jahre vertraulich bleiben müssen, sind schon heute durch Angreifer gefährdet, die verschlüsselten Datenverkehr jetzt aufzeichnen, um ihn später zu entschlüsseln; die architektonische Antwort lautet daher **Krypto-Agilität**: Führen Sie ein Inventar, wo welche Algorithmen verwendet werden, vermeiden Sie es, sie fest zu codieren, und bevorzugen Sie Bibliotheken, Protokolle und Anbieter, die wechseln können, wie es große Browser und TLS-Bibliotheken mit hybridem Post-Quanten-Schlüsselaustausch bereits begonnen haben.

Hashing, MACs, Signaturen und Verschlüsselung sind verschiedene Werkzeuge

Die Verwechslung dieser Primitive ist eine häufige Quelle von Entwurfsfehlern, etwa wenn Passwörter „verschlüsselt“ werden, sodass man sie entschlüsseln kann, oder wenn man einem einfachen Hash vertraut, um zu beweisen, dass ein Webhook von einem Zahlungsanbieter stammt. Die Tabelle stellt sie nebeneinander.

Das richtige Primitiv wählen

Werkzeug	Umkehrbar?	Schl.	Verwendung für	Beispiel
Kodierung (Base64, hex)	Ja, von jedem	Keiner	Bytes als Text darstellen. Bietet keine Sicherheit.	Ein JWT-Payload
Kryptografischer Hash (SHA-256)	Nein	Keiner	Fingerabdrücke, Integrität von Dateien gegenüber versehentlicher Änderung, inhaltsbasierte Adressierung. MD5 und SHA-1 sind hinsichtlich Kollisionsresistenz gebrochen.	Prüfsumme eines heruntergeladenen Release
Message Authentication Code (HMAC-SHA-256)	Nein	Gemeinsames Geheimnis	Nachweis, dass eine Nachricht von jemandem mit dem Schlüssel stammt und nicht verändert wurde.	Den Webhook eines Zahlungsanbieters prüfen
Digitale Signatur (Ed25519, ECDSA)	Nein (sie wird geprüft)	Privater Schlüssel signiert, öffentlicher Schlüssel prüft	Herkunftsnachweis, den jeder prüfen kann, ohne ihn fälschen zu können.	JWT-Access-Tokens, signierte Container-Images
Verschlüsselung (AES-GCM)	Ja, mit dem Schlüssel	Geheimer Schlüssel (oder ein Schlüsselpaar bei hybriden Verfahren)	Vertraulichkeit, in authentifizierten Modi zusätzlich Integrität.	Der Ausweis eines Kuriers im Speicher
Passwort-Hash (Argon2id)	Nein, und absichtlich langsam	Salt pro Passwort, optional Pepper	Passwörter so speichern, dass eine gestohlene Datenbank dem Erraten standhält.	Der Credential Store des Identity Providers

Passwörter speichern

Ein Allzweck-Hash wie SHA-256 ist darauf ausgelegt, schnell zu sein, und das ist für Passwörter genau falsch: Grafikprozessoren können Milliarden Versuche pro Sekunde gegen eine gestohlene Tabelle schneller Hashes testen. Passwort-Hashing verwendet Funktionen, die

absichtlich langsam und idealerweise speicherintensiv (memory-hard) sind, sodass jeder Versuch einen Angreifer echte Zeit und echten Speicher kostet, mit einem eindeutigen zufälligen **Salt** pro Passwort, sodass identische Passwörter unterschiedliche Hashes ergeben und vorberechnete Tabellen nutzlos sind.

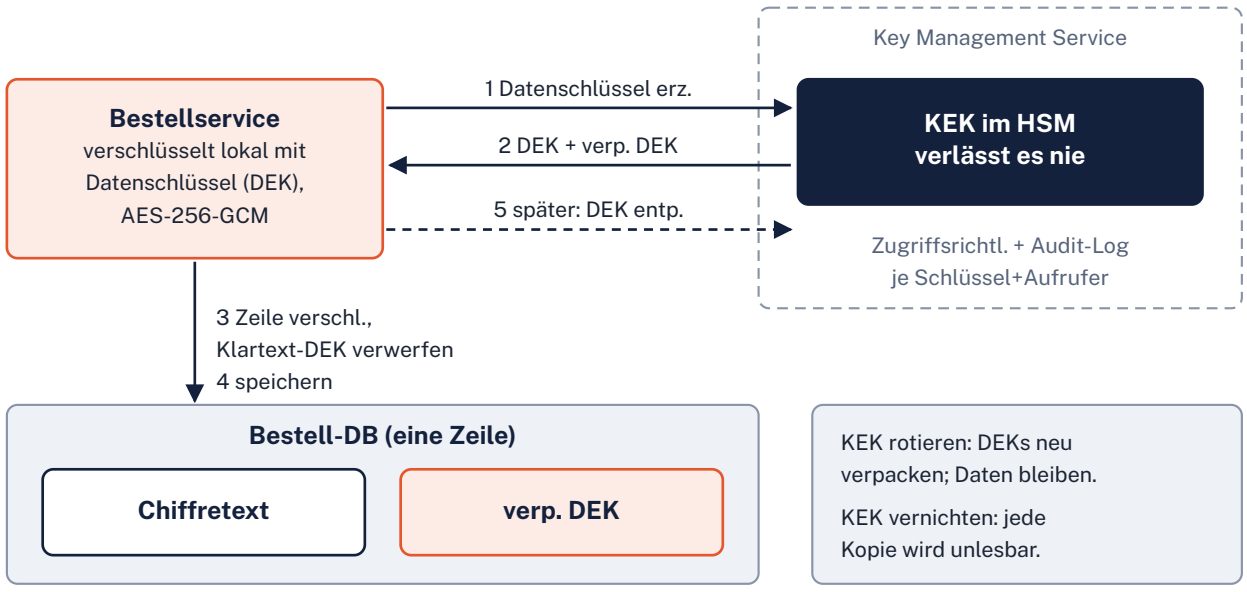
Das OWASP Password Storage Cheat Sheet gibt aktuelle Empfehlungen in der Reihenfolge ihrer Präferenz. **Argon2id** (eine Variante von Argon2, Gewinner der Password Hashing Competition 2015) mit mindestens 19 MiB Speicher, zwei Iterationen und einem Parallelitätsgrad von eins (oder einer gleichwertigen Abwägung, etwa 46 MiB mit einer Iteration). **scrypt**, wo Argon2id nicht verfügbar ist, mit einem Kostenparameter von 2^{17} (128 MiB), einer Blockgröße von 8 und einer Parallelität von 1. **bcrypt** nur für Altsysteme, mit einem Work Factor von mindestens 10, wobei zu bedenken ist, dass die meisten Implementierungen Eingaben über 72 Bytes hinaus ignorieren. Wo FIPS-140-Konformität verlangt ist, **PBKDF2** mit HMAC-SHA-256 und 600.000 Iterationen. Diese Werte steigen mit besserer Hardware, also prüfen Sie das aktuelle Cheat Sheet, statt Zahlen in ein Entwurfsdokument zu kopieren.

Daraus folgen drei Entwurfspunkte. Speichern Sie Algorithmus und Parameter mit jedem Hash (die üblichen kodierten Formate tun das), damit sich die Parameter später erhöhen lassen: Meldet sich ein Nutzer erfolgreich an, prüfen Sie gegen den alten Hash und hashen Sie sofort mit den neuen Parametern neu. Erwägen Sie einen **Pepper**, ein Geheimnis, das vor dem Hashen jedem Passwort hinzugefügt und in einem Vault oder HSM statt in der Datenbank aufbewahrt wird; das vereitelt einen Angreifer, der nur die Datenbank stiehlt. Am wichtigsten: Fragen Sie, ob Sie Passwörter überhaupt speichern sollten. Wer die Authentifizierung an einen Identity Provider delegiert und Passkeys anbietet, entfernt das am häufigsten angegriffene Geheimnis aus den meisten seiner Services.

Schlüsselverwaltung und Envelope Encryption

Schlüssel haben einen Lebenszyklus, beschrieben in NIST SP 800-57 Part 1: Erzeugung (aus einer geeigneten Zufallsquelle), Speicherung, Verteilung, Verwendung, Rotation am Ende einer festgelegten Kryptoperiode, Widerruf bei Kompromittierung und Vernichtung. All das sicher von Hand zu erledigen, ist unpraktikabel; deshalb stützen sich Architekturen auf einen Key Management Service (KMS) wie AWS KMS, Azure Key Vault oder Google Cloud KMS oder auf ein selbst betriebenes System wie HashiCorp Vault. Ein KMS hält Master Keys in Hardware-Sicherheitsmodulen (HSMs), manipulationsresistenten, nach FIPS 140 validierten Geräten, und führt Operationen mit diesen Schlüsseln auf Anfrage aus, gemäß einer Zugriffsrichtlinie, sodass Anwendungen die Master Keys nie zu sehen bekommen und jede Verwendung protokolliert wird.

Jeden Datensatz zum Verschlüsseln an den KMS zu schicken, wäre langsam und teuer, und KMS-APIs begrenzen die Größe von Anfragen. Envelope Encryption löst das. Die Anwendung fordert beim KMS einen frischen **Data Encryption Key** (DEK) an; der KMS gibt ihn zweimal zurück, im Klartext und verpackt (verschlüsselt) unter einem **Key-Encryption-Key** (KEK), der das HSM nie verlässt. Die Anwendung verschlüsselt die Daten lokal mit dem Klartext-DEK, entfernt den DEK aus dem Speicher und speichert den Chiffretext zusammen mit dem verpackten DEK. Zum Entschlüsseln sendet sie nur den kleinen verpackten DEK an den KMS, der die Berechtigung des Aufrufers prüft, die Anfrage protokolliert und den DEK zurückgibt.



Envelope Encryption: Große Datenmengen gehen nie an den KMS, und der Master Key verlässt ihn nie.

```
import os
from cryptography.hazmat.primitives.ciphers.aead import AESGCM

def encrypt_field(plaintext: bytes, kms, kek_id: str, context: bytes) → dict:
    dek = AESGCM.generate_key(bit_length=256)           # fresh data key
    nonce = os.urandom(12)                              # never reuse with this key
    ct = AESGCM(dek).encrypt(nonce, plaintext, context)  # context binds it to its row
    wrapped = kms.encrypt(key_id=kek_id, plaintext=dek)  # KEK stays inside the KMS
    return {"ct": ct, "nonce": nonce, "wrapped_dek": wrapped, "kek": kek_id}

def decrypt_field(rec: dict, kms, context: bytes) → bytes:
    dek = kms.decrypt(key_id=rec["kek"], ciphertext=rec["wrapped_dek"])  # checked,
logged
    return AESGCM(dek).decrypt(rec["nonce"], rec["ct"], context)
```

Envelope Encryption eines Felds mit der Python-Bibliothek cryptography; kms steht für einen beliebigen KMS-Client (echte KMS-APIs fassen die ersten beiden Schritte in einem Generate-Data-Key-Aufruf zusammen).

Envelope Encryption hat mehrere architektonische Vorteile. **Rotation ist günstig**: Das Rotieren eines KEK erzeugt eine neue Schlüsselversion für neue Verpackungsvorgänge, während alte Versionen zum Entschlüsseln verfügbar bleiben; vorhandene DEKs lassen sich neu verpacken, ohne die Daten anzufassen, und eine vollständige Neuverschlüsselung ist nur nötig, wenn ein DEK selbst offengelegt worden sein könnte. **Der Blast Radius ist begrenzt**: Bei einem DEK pro Datensatz, Datei oder Partition legt ein geleakter DEK wenig offen. **Funktionstrennung** wird möglich: Der Bestellservice darf seinen KEK zum Entpacken verwenden, aber weder dessen Richtlinie ändern noch ihn löschen, während Sicherheitsadministratoren Schlüssel verwalten, aber keine Daten entschlüsseln können. **Schlüsselhierarchien** lassen sich auf die Organisation abbilden: ein KEK pro Service und Datenklasse oder pro Mandant, sodass eine große Restaurantkette sogar ihren eigenen Schlüssel liefern und kontrollieren könnte. Und das Vernichten eines Schlüssels ist ein mächtiges Werkzeug: Crypto-Shredding macht jede Kopie der damit verschlüsselten Daten unlesbar, einschließlich Backups und unveränderlicher Event-Logs, was für die Löschung nach der DSGVO und für Systeme mit Event-Sourcing wichtig ist.

Der Preis ist eine neue kritische Abhängigkeit. Ist der KMS nicht verfügbar, lassen sich verschlüsselte Daten nicht lesen; wird ein Schlüssel versehentlich gelöscht, sind die Daten für immer verloren. Cloud-KMS erzwingen deshalb eine Wartefrist vor dem Löschen, und Architekten sollten das Löschen von Schlüsseln und Änderungen an Schlüsselrichtlinien als privilegierte, protokollierte Vorgänge nach dem Vier-Augen-Prinzip behandeln, entpackte DEKs kurz zwischenspeichern, wo die Latenz von KMS-Aufrufen ins Gewicht fällt, und den KMS in die Disaster-Recovery-Pläne aufnehmen.

WENN DAS NEU FÜR SIE IST

Schritt für Schritt: ein Schlüssel, der andere Schlüssel verschließt

Die Idee, die hier die meiste Sorgfalt verlangt, ist Envelope Encryption: Die Daten werden mit ihrem eigenen Datenschlüssel verschlossen, und dieser Schlüssel wird seinerseits mit einem Master Key verschlossen, der den Key Management Service (KMS) nie verlässt. Verfolgen Sie die Bankverbindung eines Restaurants für Auszahlungen auf dem Weg in den Speicher und wieder heraus.

- Um die neue Bankverbindung eines Restaurants zu speichern, fordert der Zahlungsservice beim KMS einen frischen Datenschlüssel an. Der KMS gibt ihn zweimal zurück: im Klartext und verpackt, also verschlossen unter dem Master Key für Zahlungen, der in der manipulationsresistenten Hardware des KMS bleibt.
- Der Service verschließt die Daten mit dem Klartext-Datenschlüssel auf seiner eigenen Maschine, löscht diesen Schlüssel aus dem Speicher und speichert die verschlossenen Daten zusammen mit dem verpackten Schlüssel. Die Daten gelangen nie zum KMS.
- Wer diese Zeile kopiert, hält verschlossene Daten und einen verschlossenen Schlüssel, nutzlos ohne die Berechtigung, den Master Key zu verwenden.
- Braucht der Auszahlungsjob die Daten, sendet er nur den kleinen verpackten Schlüssel an den KMS, der prüft, ob dieser Job den Master Key verwenden darf, die Anfrage protokolliert und den Klartext-Datenschlüssel zurückgibt.
- Daraus folgen zwei Vorteile. Das Rotieren des Master Keys muss die gespeicherten Daten nicht berühren: Die kleinen Datenschlüssel lassen sich einfach neu verpacken. Und das Vernichten eines Schlüssels macht jede Kopie der damit verschlüsselten Daten unlesbar, Backups eingeschlossen: Crypto-Shredding.
- Der Preis ist eine neue kritische Abhängigkeit. Ist der KMS nicht verfügbar, lassen sich die Daten nicht lesen; wird ein Schlüssel versehentlich gelöscht, sind die Daten für immer verloren.
- **Die Lehre:** Halten Sie große Datenmengen lokal und Schlüssel zentral. Jeder Datensatz kann seinen eigenen Datenschlüssel haben, während der Master Key im KMS bleibt, wo jede Verwendung geprüft und protokolliert wird.

Secrets-Management

Ein Secret ist jede Art von Zugangsdaten, die Zugriff gewährt: Datenbankpasswörter, API-Schlüssel für den Zahlungsanbieter, private Schlüssel, Signaturschlüssel, OAuth-Client-Secrets, Webhook-Secrets. Secrets-Management beginnt mit einer Regel: **Secrets gehören nie in den Quellcode**, auch nicht in Container-Images oder in Konfigurationsdateien, die in

Repositories committet werden, denn diese werden kopiert, gecacht, geforkt und von weit mehr Menschen und Systemen gelesen, als das Secret sollte. Ist ein Secret einmal in der Historie eines Repositorys, bleibt es dort, auch nachdem die Zeile gelöscht wurde.

Secrets gehören in einen Vault (HashiCorp Vault, AWS Secrets Manager, Azure Key Vault, Google Secret Manager), der sie verschlüsselt speichert, sie nur gemäß Richtlinie an authentifizierte Workloads herausgibt, jeden Zugriff protokolliert und Rotation unterstützt. Besser als langlebige Secrets zu rotieren, ist es, keine zu haben: Ein Vault kann **dynamische Secrets** ausstellen, etwa ein Datenbankkonto, das für eine einzelne Service-Instanz mit einer Lease von einer Stunde angelegt wird, und Workload Identity lässt Services sich gegenüber Cloud-APIs und untereinander ganz ohne gespeichertes Secret authentifizieren. Zur Laufzeit injizierte Secrets sollten aus Dateien oder über die API des Vaults gelesen werden statt aus breit sichtbaren Umgebungsvariablen, die gern in Crash-Dumps und Debug-Logs landen.

Vorbeugung braucht Automatisierung, weil Menschen Fehler machen. Pre-Commit-Hooks und Pipeline-Scanner (etwa gitleaks und detect-secrets) fangen Secrets ab, bevor sie gepusht werden; Repository-Hoster scannen nach bekannten Secret-Formaten, und GitHubs Push Protection blockiert Pushes, die solche enthalten. Öffentliche Repositories werden von Angreifern innerhalb von Minuten nach einem Push gescannt. Die Folgen sind gut dokumentiert: 2016 verschafften sich Angreifer Zugang zu einem privaten GitHub-Repository, das Uber-Entwickler nutzten, fanden dort einen Zugriffsschlüssel für Ubers Cloud-Speicher und luden personenbezogene Daten von rund 57 Millionen Fahrgästen und Fahrern herunter.

Checkliste: wenn ein Secret geleakt ist

Behandeln Sie jedes Secret, das in ein Repository, ein Log, ein Ticket oder einen Chat gelangt ist, als kompromittiert, ob das Repository privat ist oder nicht.

- [ ] **Das Secret sofort widerrufen oder rotieren**
  Tun Sie das zuerst. Den Commit oder die Nachricht zu löschen, hilft nicht: Klone, Forks, Caches und Scanner haben es bereits.
- [ ] **Feststellen, was es erreichen konnte**
  Listen Sie die Systeme, Daten und Berechtigungen auf, die das Secret gewährte, sowie alle weiteren Secrets, die von dort erreichbar sind.
- [ ] **Die Nutzung seit der Offenlegung prüfen**
  Prüfen Sie die Zugriffsprotokolle des Anbieters und des Vaults auf Nutzung durch unerwartete Identitäten, Adressen oder zu unerwarteten Zeiten, seit das Secret erstmals auftauchte.
- [ ] **Eindämmen und bereinigen**
  Rotieren Sie abhängige Zugangsdaten, machen Sie damit ausgestellte Sessions und Tokens ungültig und entfernen Sie das Secret dann aus Historie und Caches (nach der Rotation, nicht stattdessen).
- [ ] **Über eine Meldung entscheiden**
  Könnte auf personenbezogene Daten zugegriffen worden sein, beginnen Sie mit der Bewertung der Verletzung; nach der DSGVO beginnt die Frist für die Meldung an die Aufsichtsbehörde, sobald Ihnen die Verletzung bekannt wird.
- [ ] **Die Ursache beheben**
  Verlagern Sie das Secret in einen Vault, ersetzen Sie es durch Workload Identity oder ein dynamisches Secret und schalten Sie Secret Scanning und Push Protection ein, damit derselbe Fehler beim nächsten Mal blockiert wird.

Tokenisierung und Verkleinerung des PCI-DSS-Geltungsbereichs

Tokenisierung ersetzt einen sensiblen Wert durch ein Token, das in keiner mathematischen Beziehung zu ihm steht; der echte Wert wird in einem separaten, streng geschützten Vault aufbewahrt, und nur der Vault kann das Token zurück zuordnen. Anders als bei Verschlüsselung lässt sich ein gestohlenes Token nicht mit einem Schlüssel umkehren, weil es keinen Schlüssel gibt, nur ein Nachschlagen in einem System, das der Angreifer nicht kontrolliert.

Die wichtigste Anwendung sind Kartenzahlungen. Der Payment Card Industry Data Security Standard (PCI DSS, derzeit Version 4.0.1, veröffentlicht im Juni 2024) gilt für jede Systemkomponente, die Karteninhaberdaten speichert, verarbeitet oder überträgt oder die mit diesen Komponenten verbunden ist oder deren Sicherheit beeinflussen könnte. Eine Plattform, die Kartennummern durch ihre Server laufen lässt, holt ihr API-Gateway, ihre Services, Logs, Netzwerke und Administratoren in den Geltungsbereich, mit Hunderten von Anforderungen und einer jährlichen Prüfung. Eine Plattform, die die gehosteten Zahlungsfelder oder die Weiterleitung des Zahlungsanbieters nutzt, sodass Kartennummern direkt vom Gerät des Kunden zum Anbieter gehen, erhält nur ein Token und vielleicht die letzten vier Ziffern, und ihr PCI-DSS-Geltungsbereich schrumpft drastisch: Ein kleinerer Händler kann sich für den kürzesten Selbstbewertungsfragebogen qualifizieren, und selbst die größten Händler, die weiterhin eine jährliche Prüfung durch einen qualifizierten Assessor brauchen, haben weit weniger Systeme zu prüfen. Das ist der Entwurf im Bedrohungsmodell für den Checkout: Die am billigsten zu schützenden Kartendaten sind die, die Sie nie besitzen. Kartennummern selbst zu verschlüsseln, erreicht nicht dieselbe Verkleinerung, denn wer sie entschlüsseln kann, bleibt im Geltungsbereich.

Dieselbe Idee funktioniert für andere sensible Kennungen. Ein interner Tokenisierungsservice kann Sozialversicherungs- oder Passnummern auf der ganzen Plattform durch Tokens ersetzen, sodass nur ein kleiner, stark geschützter Service je die echten Werte sieht.

Datenklassifizierung

Verschlüsselung, Tokenisierung, Logging, Aufbewahrung und Zugriffsentscheidungen hängen alle davon ab, zu wissen, welche Daten wie sensibel sind. Ein Schema zur **Datenklassifizierung** legt einige Stufen mit zugehörigen Umgangsregeln fest, sodass Teams nicht jeden Fall von Grund auf entscheiden. Vier Stufen sind üblich; wichtig ist, dass jeder Datenbestand im Datenkatalog oder Schema der Plattform eine Stufe trägt und dass die Regeln durch Vorlagen und Prüfungen durchgesetzt werden statt durch das Gedächtnis.

Ein Datenklassifizierungsschema für die Essenslieferplattform

Niveau	Beispiele	Umgangsregeln
Öffentlich	Speisekarten, Preise, Öffnungszeiten der Restaurants	Nur Integritätsschutz; darf überall gecacht werden.
Intern	Aggregierte Umsatzzahlen, Service-Konfiguration ohne Secrets	Nur Mitarbeitende und Services; nicht in öffentlichen Logs.
Vertraulich	Namen, Adressen und Telefonnummern von Kunden, Bestellhistorien, Standorte von Kurieren	TLS und Speicherverschlüsselung; Zugriff nach Least Privilege; in Support-Tools maskiert; in der Analyse minimiert; Aufbewahrungsfristen.
Streng vertraulich	Bankverbindungen für Auszahlungen, Ausweisdokumente von Kurieren, Secrets, Zahlungstokens	Alles oben Genannte plus Verschlüsselung auf Feldebene mit eigenem Schlüssel, Zugriff nur für namentlich festgelegte Services, jeder Zugriff protokolliert.

Verschlüsselung auf Feldebene

Verschlüsselung auf Feldebene, die Schicht auf Anwendungsebene aus der ersten Tabelle, verschlüsselt einzelne Felder, bevor sie geschrieben werden, typischerweise mit Envelope Encryption und einem KEK pro Datenklasse. Ihre Stärke ist, dass alles, was der Anwendung nachgelagert ist, nur Chiffretext hält: Datenbankadministratoren, Read Replicas, Backups und das per Change Data Capture gespeiste Data Warehouse. Ein Einbruch in eines davon legt nichts Lesbares offen, und der Zugriff auf den Klartext beschränkt sich auf die Services, die den KMS mit diesem Schlüssel aufrufen können, wobei jeder Aufruf protokolliert wird.

Ihre Kosten sind real. Die Datenbank kann das verschlüsselte Feld nicht mehr indizieren, durchsuchen, sortieren oder für Joins nutzen. Gleichheitsabfragen lassen sich mit einem **Blind Index** wiederherstellen, einer separaten Spalte mit einem schlüsselbasierten Hash (HMAC) des normalisierten Werts, um den Preis, dass sichtbar wird, welche Zeilen denselben Wert teilen. Deterministische Verschlüsselung hat dasselbe Leck, während Bereichsabfragen spezielle Verfahren brauchen oder auf solchen Feldern schlicht unterbleiben sollten. Manche Datenbanken bieten clientseitige Verschlüsselung auf Feldebene oder abfragbare Verschlüsselung an, die diese Techniken bündelt. Wegen dieser Kosten sollte Verschlüsselung auf Feldebene streng vertraulichen Daten vorbehalten bleiben, deren Felder ohnehin selten abgefragt werden.

AUSGEARBEITETES BEISPIEL

Ausgearbeitetes Beispiel: was verschlüsseln, wo und mit welchen Schlüsseln

Die Architektin der Essenslieferplattform prüft die wichtigsten Datenbestände anhand des Klassifizierungsschemas. Jeder Service nutzt bereits TLS (intern Mutual TLS) und verschlüsselten Speicher mit einem KMS-Schlüssel pro Service; die Frage ist also, was mehr braucht.

Kartennummern	Werden nie gehalten. Gehostete Zahlungsfelder senden sie an den Zahlungsanbieter; der Zahlungsservice speichert das Token des Anbieters, die Kartenmarke und die letzten vier Ziffern. Der PCI-DSS-Geltungsbereich schrumpft auf die Checkout-Integration mit dem Anbieter.
Passwörter der Kunden	Werden von den Produktservices nicht gespeichert. Der Identity Provider bietet Passkeys an; verbleibende Passwörter werden mit Argon2id nach den OWASP-Parametern gehasht, mit einem Pepper in einem HSM.
Adressen und Telefonnummern (vertraulich)	Speicherverschlüsselung mit dem KEK des Bestellservice; Konten nach Least Privilege; im Support-Tool maskiert; das Data Warehouse erhält nur Postleitzahlbezirke, keine vollständigen Adressen.
Bankverbindungen der Restaurants für Auszahlungen (streng vertraulich)	Envelope Encryption auf Feldebene unter einem eigenen KEK des Zahlungsservice; nur der Auszahlungsjob darf entpacken; Änderungen erfordern eine erneute Authentifizierung und benachrichtigen den Restaurantbesitzer, was neben der Vertraulichkeit auch die Integrität schützt.
Ausweisdokumente von Kurieren (streng vertraulich)	Als Objekte gespeichert, jedes mit eigenem DEK, verpackt unter einem Onboarding-KEK; nur der Verifizierungsservice kann entschlüsseln; jeder Zugriff wird protokolliert und monatlich geprüft; Löschung nach Ablauf der gesetzlichen Aufbewahrungsfrist.
Personenbezogene Daten in Events und im Event-Log	Events tragen Kunden-IDs statt Adressen; die wenigen personenbezogenen Felder, die mitreisen müssen, werden mit einem Schlüssel pro Kunde verschlüsselt, sodass die Vernichtung des Schlüssels sie aus dem aufbewahrten Log löscht.
Backups	Verschlüsselt mit einem separaten KEK in einem separaten Cloud-Konto, sodass kompromittierte Produktionszugangsdaten die Backups nicht offenlegen; Wiederherstellungen werden vierteljährlich getestet.
Service-Secrets	Im Vault; Datenbanken nutzen dynamische Zugangsdaten mit Leases von einer Stunde; der API-Schlüssel des Zahlungsanbieters ist nur für die Workload Identity des Zahlungsservice lesbar.

Etwa zehn KEKs, jeder im Besitz eines Service und jährlich automatisch rotiert, decken die ganze Plattform ab; die Hebelwirkung liegt in den Entscheidungen, nicht in der Zahl der Schlüssel.

Zwei Entscheidungen entfernen Daten, statt sie zu schützen (Kartennummern und vollständige Adressen in der Analyse), und das ist immer die stärkste Option.

Der Entwurf wird als ein ADR pro streng vertraulichem Datenbestand und als Datenschutzkonzept in der Architekturbeschreibung festgehalten.

PRÜFEN SIE IHR VERSTÄNDNIS

▼ **Vor welchen Parteien schützt Transparent Database Encryption nicht?**

Vor jedem, der die Datenbank über ein gültiges Konto abfragen kann, einschließlich der Anwendung, eines Angreifers per SQL-Injection und der Datenbankadministratoren, weil die Datenbank die Daten für sie entschlüsselt.

▼ **Warum ist das Rotieren eines Key-Encryption-Keys bei Envelope Encryption günstig?**

Nur die kleinen verpackten Datenschlüssel müssen unter der neuen Schlüsselversion neu verpackt werden; die großen Datenmengen bleiben unter ihren unveränderten Datenschlüsseln verschlüsselt.

▼ **Ein Entwickler schlägt vor, Passwörter mit AES verschlüsselt zu speichern, damit der Support Nutzern ihr Passwort nennen kann. Was ist daran falsch?**

Passwörter müssen mit einem langsamen, gesalzenen Einweg-Passwort-Hash gespeichert werden, damit niemand sie wiederherstellen kann, weder Mitarbeitende noch wer den Schlüssel stiehlt; eine Wiederherstellung sollte zurücksetzen, nicht offenlegen.

▼ **Warum verkleinert die Nutzung der gehosteten Felder eines Zahlungsanbieters den PCI-DSS-Geltungsbereich stärker als die eigene Verschlüsselung von Kartennummern?**

Mit gehosteten Feldern erreichen Kartennummern die Systeme der Plattform nie, also fallen diese nicht in den Geltungsbereich; verschlüsselte Kartennummern, die die Plattform entschlüsseln kann, halten ihre Systeme im Geltungsbereich.

▼ **Probieren Sie es aus: Die Plattform nutzt bereits TLS und verschlüsselten Speicher. Entscheiden Sie, was für Kartennummern, Passwörter und Allergiehinweise der Kunden noch zu tun ist. Nennen Sie jeweils den Schutz und die Bedrohung, gegen die er wirkt. Gehen Sie davon aus, dass die Klassifizierung Allergiehinweise als sensibel einstuft.**

Kartennummern: halten Sie sie nie vor; verwenden Sie gehostete Zahlungsfelder und das Token des Anbieters, was den PCI-DSS-Geltungsbereich verkleinert. Passwörter: keine Verschlüsselung, sondern ein langsamer gesalzener Hash wie Argon2id oder Passkeys über einen Identity Provider, sodass eine gestohlene Tabelle wenig nützt. Allergiehinweise: Verschlüsselung auf Feldebene mit Envelope-Schlüsseln in einem KMS, zum Schutz vor Administratoren und nachgelagerten Kopien. Das kostet Durchsuchbarkeit und schafft eine Abhängigkeit vom KMS, behalten Sie sie also sensiblen Feldern vor.

QUELLEN

- OWASP Password Storage Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html
- OWASP Secrets Management Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Secrets_Management_Cheat_Sheet.html
- NIST SP 800-57 Part 1 Rev. 5: Recommendation for Key Management: https://csrc.nist.gov/pubs/sp/800/57/pt1/r5/final
- Google Cloud KMS: Envelope encryption: https://cloud.google.com/kms/docs/envelope-encryption
- PCI Security Standards Council: https://www.pcisecuritystandards.org/

Sichere Kommunikation und Zero Trust

BEVOR SIE DAS LESEN

Dieser Abschnitt baut auf:

- 0.4 Wie Computer kommunizieren
- 8.3 Identität und Zugriff
- 8.4 Daten schützen

Begriffe von früher, die er verwendet: TLS und HTTPS, OAuth 2.0, Workload Identity, Fallacies of Distributed Computing, API-Gateway

DIE IDEE IN KÜRZE

Stellen Sie sich ein Hotel vor, in dem die Rezeption Ihre Identität einmal prüft und Ihnen dann einen Schlüssel aushändigt, der jedes Zimmer im Gebäude öffnet. Kommt ein Dieb an der Rezeption vorbei, steht ihm jedes Zimmer offen. Ein besseres Hotel gibt Ihnen eine Schlüsselkarte, die nur Ihr Zimmer öffnet, nur für die gebuchten Nächte, und die jede Tür protokolliert, die sie öffnet. Zero Trust überträgt die zweite Idee auf Software: Keine Anfrage gilt als vertrauenswürdig, nur weil sie aus dem Firmennetzwerk kommt. Jede Person und jedes Programm muss jedes Mal nachweisen, wer es ist, und genau diese Handlung ausführen dürfen.

WAS SIE WISSEN MÜSSEN

- Das Perimetermodell vertraut allem innerhalb des Netzwerks, sodass ein einziges gestohlenes Zugangsdatum oder ein kompromittierter Rechner laterale Bewegung ermöglicht; Cloud, Mobilgeräte und Remote-Arbeit haben zudem jeden einzelnen zu verteidigenden Perimeter aufgelöst.
- NIST SP 800-207 (August 2020) definiert Zero Trust als den Verzicht auf jedes implizite Vertrauen aufgrund von Netzwerkstandort oder Eigentümerschaft, mit sieben Grundsätzen, darunter die Absicherung jeder Kommunikation, Zugriff pro Sitzung und eine dynamische, strikt durchgesetzte Richtlinie.
- Ein Policy Enforcement Point vor jeder Ressource fragt einen Policy Decision Point (Policy Engine plus Policy Administrator), der anhand von Identität, Geräte- oder Workload-Zustand, Bedrohungsdaten und Richtlinie entscheidet.

- Googles BeyondCorp (2014) wandte Zero Trust auf den Zugriff der Belegschaft an, mit Geräteinventar, Access Proxy und Vertrauensstufen, und schaffte das privilegierte Büronetzwerk und das VPN ab.
- Mutual TLS gibt jedem Service-Aufruf Verschlüsselung und eine verifizierte Identität an beiden Enden; SPIFFE standardisiert kurzlebige Workload-Identitäten, und Service Meshes automatisieren mTLS und Richtlinien pro Aufruf, zu echten Betriebskosten.
- Zertifikate müssen automatisch ausgestellt und erneuert werden (ACME, RFC 8555), zumal die Laufzeit öffentlicher TLS-Zertifikate 2026 auf 200 Tage, 2027 auf 100 Tage und 2029 auf 47 Tage sinkt.
- API-Gateways authentifizieren Tokens, begrenzen die Anfragerate und validieren Eingaben, doch die Autorisierung auf Objektebene muss weiterhin in dem Service stattfinden, dem die Daten gehören.

Warum der Netzwerkperimeter nicht mehr funktioniert

Jahrzehntelang schützten die meisten Organisationen ihre Systeme wie eine Burg mit Wassergraben. Eine Firewall trennte das vertrauenswürdige Innere vom nicht vertrauenswürdigen Internet, die Belegschaft verband sich über ein virtuelles privates Netzwerk (VPN) mit dem Inneren, und alles, was schon drinnen war, galt als freundlich. Services im Rechenzentrum riefen einander ohne Authentifizierung über einfaches HTTP auf, weil das Netzwerk selbst als Berechtigungsnachweis galt. Kapitel 6 nannte „das Netzwerk ist sicher“ unter den Fallacies of Distributed Computing; das Modell der Perimetersicherheit ist dieser Trugschluss, zur Architektur gemacht.

Drei Dinge haben es zu Fall gebracht. Erstens wurde das Innere zu groß, um es zu verteidigen: Laptops reisen, die Belegschaft arbeitet von zu Hause, Partner verbinden sich, und eine einzige Phishing-E-Mail oder ein gestohlenes VPN-Passwort bringt einen Angreifer in den Burggraben. Ist er erst drinnen, bewegt er sich seitwärts von Rechner zu Rechner; das nennt man laterale Bewegung (Lateral Movement). Beim Angriff auf den US-Händler Target im Jahr 2013 nutzten die Angreifer zunächst Netzwerk-Zugangsdaten, die einem Dienstleister für Heizung und Lüftung gestohlen worden waren, gelangten von dort zu den Kassenterminals und erbeuteten Daten zu rund 40 Millionen Karten. Zweitens löste sich das Innere auf: Workloads laufen heute über mehrere Cloud-Konten, SaaS-Produkte und APIs Dritter verteilt, sodass es keine einzelne Mauer mehr gibt, auf die man eine Firewall setzen könnte. Drittens wurde das Innere dynamisch: Container entstehen und verschwinden jede Minute, und IP-Adressen werden wiederverwendet, sodass Regeln wie *erlaube 10.0.4.0/24 den Zugriff auf 10.0.7.12* nichts Stabiles mehr beschreiben.

Die Idee, den Perimeter aufzugeben, ist älter als ihr Name. Das Jericho Forum, eine Gruppe von Sicherheitsverantwortlichen aus Unternehmen, warb ab 2004 für *De-Perimeterisierung*; John Kindervag, damals Analyst bei Forrester Research, prägte um 2010 den Begriff *Zero Trust*; und Googles BeyondCorp-Veröffentlichungen ab 2014 zeigten, dass es im großen Maßstab funktioniert. Das Motto lautet *never trust, always verify* (nie vertrauen, immer prüfen): Keine Anfrage gilt als vertrauenswürdig, nur weil sie von einem bestimmten Ort kommt.

WENN DAS NEU FÜR SIE IST

Wenn das neu für Sie ist

Denken Sie an ein Bürogebäude. Der alte Ansatz prüft Ihren Ausweis an der Eingangstür und lässt Sie dann in jeden Raum. Zero Trust prüft Ihren Ausweis an jeder Tür, die Sie öffnen, und fragt, ob Sie gerade jetzt dort sein sollten. Das Gebäude hat kein einzelnes vertrauenswürdiges Inneres mehr, sodass ein Dieb, der hinter jemandem durch die Eingangstür schlüpft, trotzdem nicht in die Finanzabteilung gelangt.

Der Rest dieses Abschnitts verwendet einige Begriffe aus früheren Abschnitten: TLS und Zertifikate (Abschnitt 0.4), Tokens, OAuth 2.0 und Workload Identity (Abschnitt 8.3) sowie Verschlüsselung und Schlüssel (Abschnitt 8.4).

Zero Trust nach der Definition des NIST

Der Referenztext ist die NIST Special Publication 800-207, *Zero Trust Architecture*, veröffentlicht im August 2020 (Rose, Borchert, Mitchell und Connelly). Sie beschreibt Zero Trust als eine Reihe von Prinzipien, die die Verteidigung von statischen, netzwerkbasierten Perimetern hin zu Nutzern, Assets und Ressourcen verlagern, und eine Zero-Trust-Architektur als den Plan einer Organisation, diese Prinzipien anzuwenden. Ihre zentrale Aussage lautet, dass einem Asset oder Benutzerkonto kein implizites Vertrauen allein aufgrund seines physischen oder logischen Netzwerkstandorts oder seiner Eigentümerschaft gewährt wird. Authentifizierung und Autorisierung sind eigenständige Funktionen, die ausgeführt werden, bevor eine Sitzung zu einer Ressource aufgebaut wird.

Die Publikation nennt sieben Grundsätze. Es lohnt sich, sie im Wortlaut zu kennen, denn viele Produkte behaupten, Zero Trust zu sein, erfüllen aber nur einen oder zwei davon.

Die sieben Grundsätze von NIST SP 800-207, am Beispiel der Essenslieferplattform

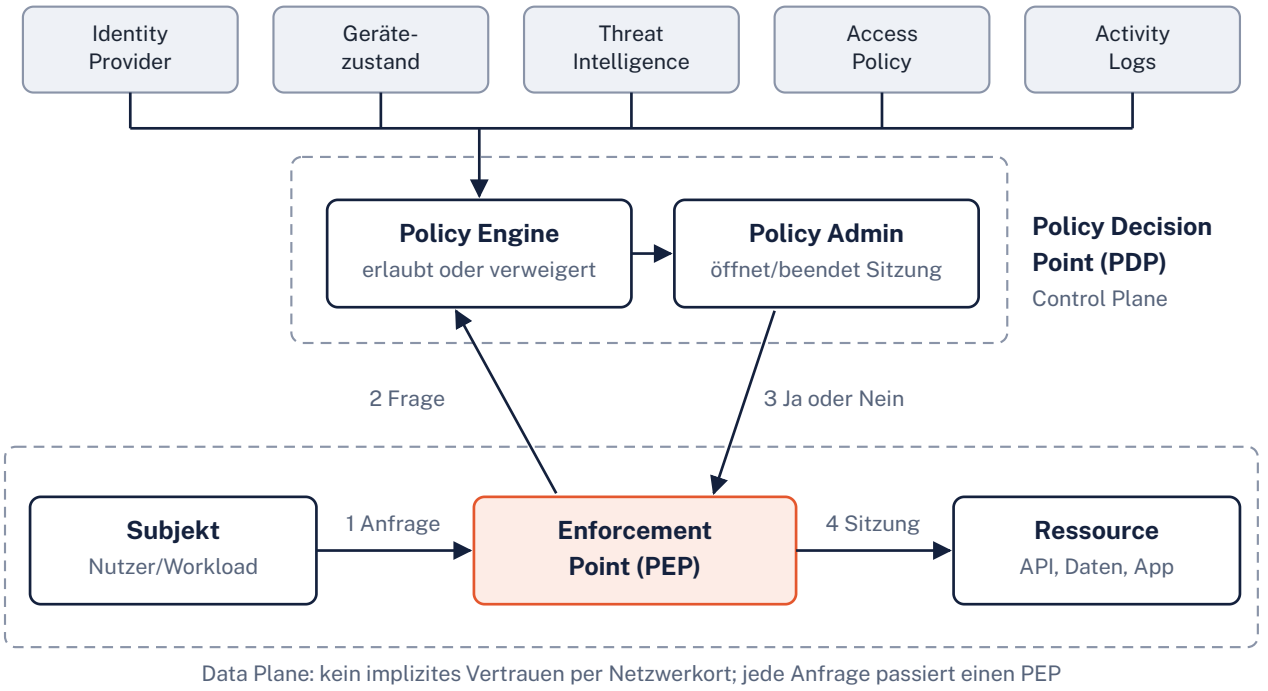
Grundsatz (sinngemäß)	Was er bedeutet	Beispiel Essenslieferung
1. Alle Datenquellen und Rechendienste sind Ressourcen	Ein Laptop, ein Container, ein SaaS-Tool und eine Datenbank sind allesamt schützenswert, nicht nur die Server im Rechenzentrum.	Das Backend der Kurier-App, die API für die Restaurant-Tablets und der Analytics-Bucket sind jeweils Ressourcen mit eigener Richtlinie.
2. Jede Kommunikation wird unabhängig vom Netzwerkstandort abgesichert	Datenverkehr innerhalb des Rechenzentrums erhält dieselbe Verschlüsselung und Authentifizierung wie Datenverkehr über das Internet.	Bestellungen ruft Zahlungen über Mutual TLS auf, obwohl beide im selben Cluster laufen.
3. Zugriff wird pro Sitzung gewährt	Vertrauen wird für jede Sitzung mit den geringsten nötigen Rechten bewertet, und der Zugriff auf eine Ressource gewährt nichts darüber hinaus.	Die Sitzung eines Support-Mitarbeiters kann die Bestellhistorie eines Kunden lesen; die Erstattungs-API erreicht sie damit nicht.
4. Über den Zugriff entscheidet eine dynamische Richtlinie	Die Entscheidung nutzt die Identität, den Zustand des Geräts oder Workloads und weitere Attribute wie Uhrzeit, Ort und Verhalten.	Eine Erstattung über einem festgelegten Betrag von einem nicht verwalteten Laptop wird abgelehnt; dieselbe Anfrage von einem verwalteten Gerät mit frischer MFA wird erlaubt.
5. Das Unternehmen überwacht die Integrität und den Sicherheitszustand aller Assets	Kein Gerät und kein Workload gilt von vornherein als vertrauenswürdig; Patchstand, Konfiguration und Integrität werden laufend gemessen.	Ein Container-Image ohne gültige Signatur oder mit einer kritischen, ungepatchten CVE wird nicht in den Cluster gelassen.
6. Authentifizierung und Autorisierung sind dynamisch und werden vor dem Zugriff strikt durchgesetzt	Die Identität wird vor jedem Zugriff geprüft und bei langen Sitzungen erneut geprüft; Zugangsdaten sind kurzlebig und werden rotiert.	Service-Zertifikate gelten wenige Stunden und werden automatisch erneuert; Nutzersitzungen werden neu bewertet, wenn sich das Risiko ändert.
7. Das Unternehmen sammelt so viele Informationen wie möglich über den Zustand von Assets, Netzwerk und Kommunikation	Telemetrie fließt in die Richtlinie zurück, damit sich die Entscheidungen mit der Zeit verbessern.	Zugriffslogs von Gateway, Mesh und Identity Provider fließen in die Plattform für das Sicherheitsmonitoring.

Entscheidungs- und Durchsetzungspunkte

SP 800-207 beschreibt die Mechanik mit wenigen logischen Komponenten. Ein Policy Enforcement Point (PEP) sitzt vor einer Ressource und lässt Datenverkehr nur durch, wenn er dazu angewiesen wird: in der Praxis ein identitätsbewusster Proxy, ein API-Gateway, ein Sidecar eines Service Meshes oder ein Datenbank-Proxy. Der Policy Decision Point (PDP) trifft die Entscheidung. Das NIST teilt ihn in eine Policy Engine, die die Anfrage anhand von Richtlinie und Eingaben bewertet und entscheidet, und einen Policy Administrator, der die Entscheidung umsetzt, indem er den Pfad öffnet oder schließt, etwa durch Ausgabe eines kurzlebigen Sitzungstokens oder durch Konfiguration des PEP. Zu den Eingaben gehören der Identity Provider, der Zustand von Geräten und Workloads (Patchstand, Konfiguration, Integrität), Threat Intelligence, die Zugriffsrichtlinien selbst und Aktivitätslogs.

Entscheidung und Durchsetzung zu trennen ist derselbe Schritt wie die Trennung von Richtlinie und Mechanismus in Betriebssystemen, und er lohnt sich aus demselben Grund: Richtlinien lassen sich einmal schreiben, zentral prüfen, testen und ändern, während die Durchsetzung an jeden Ort verteilt ist, an dem Datenverkehr fließt. Die Policy Engines aus Abschnitt 8.3, etwa Open Policy Agent, sind oft die Engine hinter einem PDP. Die Designfrage für Architekten lautet, wo die Durchsetzungspunkte sitzen müssen, damit es keinen Weg an ihnen vorbei gibt, und wie die Entscheidung schnell und verfügbar bleibt, denn ein PDP im kritischen Pfad jeder Anfrage ist nun eine Abhängigkeit mit eigenem Budget für Verfügbarkeit und Latenz. Übliche Antworten sind, Entscheidungen für einige Sekunden zu cachen, kompilierte Richtlinien an den PEP zu verteilen, damit er lokal entscheiden kann, und vorab festzulegen, was geschieht, wenn der PDP nicht erreichbar ist (normalerweise Ablehnung, mit einer kleinen Allow-List für Health Checks).

Die logischen Komponenten einer Zero-Trust-Architektur



Anfragen laufen nur über einen Durchsetzungspunkt; entschieden wird in einer getrennten Control Plane, die Identitäts-, Zustands-, Bedrohungs- und Richtliniendaten erhält.

BeyondCorp: Zero Trust für Menschen

Google begann nach dem Angriff von 2009, bekannt als Operation Aurora, sich von seinem privilegierten Firmennetzwerk zu lösen, und beschrieb das Ergebnis in *BeyondCorp: A New Approach to Enterprise Security* (Rory Ward und Betsy Beyer, ;login:, Dezember 2014), gefolgt von mehreren weiteren Veröffentlichungen. Das Design hat drei Hauptteile. Jedes Gerät ist in einem Geräteinventar erfasst und wird über ein Zertifikat identifiziert. Ein aus dem Internet erreichbarer Access Proxy steht vor jeder internen Anwendung; Beschäftigte erreichen Anwendungen über ihn aus jedem Netzwerk, auch aus dem Büronetzwerk, das keine besonderen Rechte erhält. Eine Komponente zur Vertrauensbewertung verknüpft, wer der Nutzer ist, mit dem Zustand des Geräts, um eine Vertrauensstufe zuzuweisen, und jede Anwendung legt die Mindeststufe fest, die sie verlangt.

Die Folge, die viele überraschte, war der Wegfall des VPN: Wer im Café sitzt, durchläuft dieselben Prüfungen wie jemand am Schreibtisch im Büro. Die Idee wird heute als Produkte unter den Namen Identity-Aware Proxy und Zero Trust Network Access (ZTNA) angeboten. Das Zero Trust Maturity Model der US-amerikanischen Cybersecurity and Infrastructure Security Agency (Version 2.0, April 2023) ist ein nützliches Werkzeug zur Selbsteinschätzung: Es bewertet fünf Säulen (Identität, Geräte, Netzwerke, Anwendungen und Workloads sowie Daten) von traditionell bis optimal, mit Sichtbarkeit, Automatisierung und Governance quer über alle.

Mutual TLS, Zertifikate und Workload Identity

Zero Trust zwischen Services braucht bei jedem Aufruf zweierlei: Verschlüsselung und eine starke Identität für beide Enden. Transport Layer Security leistet beides. TLS 1.3 (RFC 8446, August 2018) schließt seinen Handshake in einem Round Trip ab: Der Client sendet die unterstützten Algorithmen und einen Key Share, der Server antwortet mit seinem Key Share und seinem Zertifikat, und beide leiten mit ephemerem Diffie-Hellman Sitzungsschlüssel ab, sodass jede Sitzung Forward Secrecy hat (ein gestohlener Serverschlüssel kann aufgezeichneten früheren Datenverkehr nicht entschlüsseln). TLS 1.3 entfernte die schwachen Optionen früherer Versionen, darunter RSA-Schlüsselaustausch, Chiffren im CBC-Modus, RC4, SHA-1-Signaturen, Kompression und Renegotiation; TLS 1.0 und 1.1 wurden 2021 durch RFC 8996 formell für veraltet erklärt. Ein optionaler Zero-Round-Trip-Modus (0-RTT) lässt einen wiederkehrenden Client sofort Daten senden, doch diese frühen Daten kann ein Angreifer wiederholt einspielen, daher müssen sie auf idempotente Anfragen beschränkt bleiben. Große Browser und Cloud-Anbieter verwenden inzwischen auch einen hybriden Post-Quanten-Schlüsselaustausch, der X25519 mit ML-KEM kombiniert (2024 vom NIST als FIPS 203 standardisiert) und heute aufgezeichneten Datenverkehr gegen einen künftigen Quantencomputer schützt.

Ein Zertifikat bindet einen öffentlichen Schlüssel an einen Namen und ist von einer Zertifizierungsstelle (CA) signiert; der Client prüft die Signaturkette bis zu einer Wurzel, der er bereits vertraut. Dieses System aus CAs, Zertifikaten, Trust Stores und Widerruf ist eine Public-Key-Infrastruktur (PKI). Öffentliche Zertifikate werden zudem in Certificate-Transparency-Logs eingetragen, sodass ein fälschlich ausgestelltes Zertifikat für Ihre Domain auffallen kann. Gewöhnliches HTTPS authentifiziert nur den Server. Bei Mutual TLS (mTLS) legt auch der Client ein Zertifikat vor, sodass der Server erfährt, welcher Workload aufruft. Interne Zertifikate stammen von einer privaten CA, die die Organisation selbst betreibt, meist innerhalb ihrer Secrets- oder Mesh-Plattform, weil öffentliche CAs keine Zertifikate für interne Namen ausstellen.

Welcher Name gehört ins Zertifikat? Hostnamen und IP-Adressen sind schlechte Identitäten für Container. SPIFFE (Secure Production Identity Framework for Everyone, ein graduiertes Projekt der Cloud Native Computing Foundation) standardisiert Workload Identity: Jeder Workload erhält eine URI wie `spiffe://fooddelivery.example/ns/prod/sa/orders` und ein kurzlebiges SPIFFE Verifiable Identity Document, entweder ein X.509-Zertifikat für mTLS oder ein JWT. Die Referenzimplementierung SPIRE attestiert einen Workload (prüft, welcher Kubernetes-Service-Account oder welche Cloud-Instanz er tatsächlich ist), bevor sie das Dokument ausstellt, und rotiert es automatisch, typischerweise etwa stündlich. Autorisierungsregeln können dann *orders darf payments aufrufen* sagen, statt Adressen zu nennen.

Service Meshes

mTLS, Zertifikatsrotation, Wiederholungen und Autorisierung pro Aufruf in jedem Service und in jeder Sprache umzusetzen ist teuer und uneinheitlich. Ein Service Mesh verlagert diese Arbeit in die Infrastruktur. Im klassischen Design erhält jede Service-Instanz einen Sidecar-Proxy (Envoy in Istio, ein Rust-Mikroproxy in Linkerd), der ihren Datenverkehr abfängt; eine Control Plane stellt Identitäten aus, verteilt Richtlinien und sammelt Telemetrie. Neuere Designs ohne Sidecar, etwa der Ambient Mode von Istio, übertragen mTLS einem gemeinsamen Proxy pro Knoten und fügen nur dort einen separaten Proxy hinzu, wo Richtlinien auf HTTP-Ebene nötig sind. NIST SP 800-207A (September 2023) beschreibt dieses Muster als den Weg, Zero Trust auf Cloud-native Anwendungen anzuwenden: identitätsbasierte Segmentierung, bei der das Mesh die Richtlinien bei jedem Aufruf durchsetzt.

Ein Mesh ist eine bedeutende Plattformentscheidung. Es fügt pro Hop Latenz hinzu (meist gering, aber beim p99 messbar), Speicher pro Pod, eine Control Plane, die selbst hochverfügbar und abgesichert sein muss, und eine Lernkurve für jedes Team. Außerdem schützt es nur Datenverkehr, der durch das Mesh läuft: Ein Job, der sich direkt mit der Datenbank verbindet, umgeht es. Kleine Systeme mit einer Handvoll Services fahren oft besser mit mTLS über eine Bibliothek oder die Load Balancer der Plattform. Die entscheidenden Fragen sind, wie viele Services und Sprachen Sie haben und ob es ein Plattformteam gibt, das das Mesh betreibt (siehe Abschnitt 10.7).

Zertifikate automatisieren

Zertifikate laufen ab, und abgelaufene Zertifikate gehören zu den häufigsten Ursachen vermeidbarer Ausfälle. Am 6. Dezember 2018 ließ ein abgelaufenes Zertifikat in Ericsson-Software, die das britische Mobilfunknetz O2 nutzte, rund 32 Millionen Kunden den größten Teil eines Tages ohne Datendienst. Das Gegenmittel ist Automatisierung. Das ACME-Protokoll (Automatic Certificate Management Environment, RFC 8555, 2019), entwickelt für die Zertifizierungsstelle Let’s Encrypt, lässt Software die Kontrolle über eine Domain nachweisen (indem sie ein Token über HTTP ausliefert, einen DNS-Eintrag anlegt oder eine TLS-Challenge beantwortet), ein Zertifikat beziehen und es ohne menschliches Zutun erneuern. Let’s Encrypt, eine kostenlose, gemeinnützige CA, die 2015 mit der Ausstellung von Zertifikaten begann, machte kurzlebige, automatisch erneuerte Zertifikate zur Norm; Tools wie cert-manager tun dasselbe in Kubernetes, und jeder große Cloud-Load-Balancer kann Zertifikate für Sie verwalten.

Automatisierung ist nicht mehr optional. Nach dem Ballot SC-081 des CA/Browser Forum sank die maximale Laufzeit öffentlich vertrauenswürdiger TLS-Zertifikate für ab dem 15. März 2026 ausgestellte Zertifikate auf 200 Tage und sinkt ab dem 15. März 2027 auf 100 Tage und ab dem 15. März 2029 auf 47 Tage. Kurze Laufzeiten begrenzen den Schaden eines gestohlenen

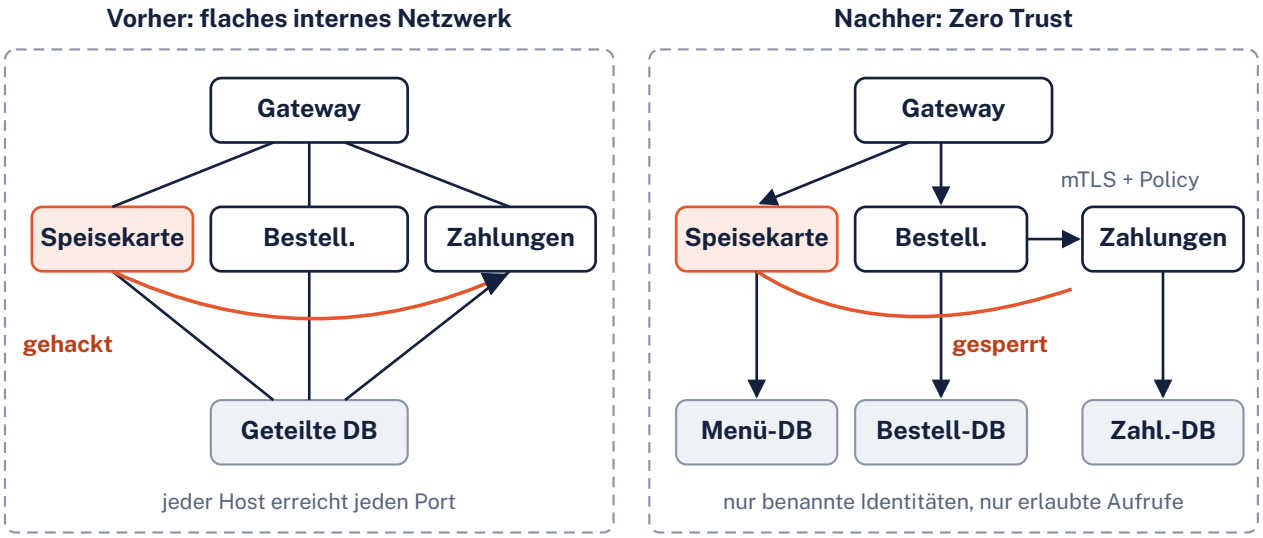
Schlüssels und verringern die Abhängigkeit vom Widerruf, der schon immer schlecht funktioniert hat. Architektonisch bedeutet das: Jedes Zertifikat, ob öffentlich oder intern, braucht einen Verantwortlichen, eine automatisierte Erneuerung, eine Überwachung der Ablaufdaten mit Alarmen lange vor der Frist und einen eingeübten manuellen Notfallweg.

Segmentierung und Mikrosegmentierung

Zero Trust bedeutet nicht, dass das Netzwerk keine Rolle mehr spielt; es bedeutet, dass das Netzwerk nicht mehr die einzige Kontrolle ist. Netzwerksegmentierung teilt die Infrastruktur in Zonen mit kontrollierten Pfaden dazwischen: öffentliche Subnetze für Load Balancer, private Subnetze für Services, isolierte Subnetze für Datenbanken und getrennte Cloud-Konten für Produktion, Staging und Sicherheitswerkzeuge. Mikrosegmentierung verkleinert die Zonen auf einzelne Workloads, sodass jeder Service Datenverkehr nur von den Aufrufern empfangen darf, die er braucht, und nur an seine deklarierten Abhängigkeiten senden darf.

In Kubernetes können alle Pods alle anderen Pods erreichen, solange keine NetworkPolicy-Ressourcen sie einschränken (und solche Richtlinien wirken nur, wenn das Netzwerk-Plugin des Clusters sie durchsetzt); der erste Schritt ist daher eine Default-Deny-Richtlinie pro Namespace, gefolgt von expliziten Freigaben. Die Kontrolle des ausgehenden Datenverkehrs (Egress) ist ebenso wichtig wie die des eingehenden: Ein kompromittierter Service, der keine Verbindungen ins Internet öffnen kann, kann gestohlene Daten nicht ohne Weiteres hinausschicken oder weitere Werkzeuge herunterladen. Security Groups in der Cloud, Kubernetes-Network-Policies und Autorisierungsrichtlinien des Meshes arbeiten auf verschiedenen Ebenen (Adressen und Ports, Pods und Labels, Workload-Identitäten und HTTP-Methoden) und werden am besten zusammen eingesetzt: Die äußeren Ebenen bieten groben Schutz, der auch eine Fehlkonfiguration des Meshes übersteht, die innere Ebene drückt die Absicht präzise aus. Das ist Defense in Depth (Abschnitt 8.1), angewandt auf das Netzwerk.

Vom flachen Netzwerk zur identitätsbasierten Segmentierung



Zero Trust verhindert nicht, dass ein Service kompromittiert wird; es begrenzt, was der kompromittierte Service erreichen kann.

Das API-Gateway absichern

Das API-Gateway ist der wichtigste Durchsetzungspunkt zwischen dem Internet und den Services und hat daher mehrere Aufgaben. **Authentifizierung:** TLS terminieren, dann jedes Access Token validieren (Signatur mit fest vorgegebenem Algorithmus, Aussteller, Zielgruppe, Ablaufzeit und Not-before-Zeit), Anfragen ohne Token ablehnen und an die Services eine verifizierte Identität weitergeben, niemals die rohen Angaben des Clients. **Rate Limiting:** Anfragen pro Client, pro Nutzer und pro Route begrenzen, meist mit einem Token Bucket, und mit HTTP 429 samt `Retry-After`-Header antworten; eigene, strengere Limits schützen Login, Passwort-Zurücksetzung und das Einlösen von Gutscheinen vor Credential Stuffing und Enumeration. **Eingabevalidierung:** maximale Anfragegrößen und Header-Anzahlen durchsetzen, nur die deklarierten Content-Types akzeptieren und Bodys gegen das OpenAPI-Schema validieren, damit fehlerhafte Anfragen nie einen Service erreichen. Eine Web Application Firewall kann generische Angriffssignaturen ergänzen, ist aber eine Ergänzung, kein Ersatz für korrekten Code.

Kennen Sie die Grenzen eines Gateways. Es sieht die Route und das Token, aber nicht, ob Bestellung 81723 dem Kunden gehört, der sie anfordert. Broken Object Level Authorization, bei der ein Nutzer eine Kennung in der URL ändert und die Daten eines anderen liest, stand sowohl in der Ausgabe 2019 als auch in der Ausgabe 2023 der OWASP API Security Top 10 an erster Stelle, und nur der Service, dem die Daten gehören, kann sie verhindern. Services müssen daher jede Anfrage selbst autorisieren, mit der verifizierten Identität, die das Gateway weiterreicht. Ruft ein Service im Auftrag eines Nutzers einen anderen auf, sollte er ein Token weitergeben, das sowohl den Nutzer als auch den aufrufenden Service nennt, etwa per OAuth

2.0 Token Exchange (RFC 8693) bezogen, statt seines eigenen allmächtigen Berechtigungsnachweises; sonst macht ein Fehler in einem Service ihn zu einem Confused Deputy, der für jeden handeln kann.

AUSGEARBEITETES BEISPIEL

Ausgearbeitetes Beispiel: die Essenslieferplattform vom flachen Netzwerk zu Zero Trust überführen

Die Plattform betreibt etwa zwanzig Services in einem Kubernetes-Cluster und einem Cloud-Konto. Die Services rufen einander über einfaches HTTP auf, sechs Services teilen sich einen Datenbanknutzer, die Belegschaft erreicht die Admin-Konsole über ein VPN, und das Bedrohungsmodell aus Abschnitt 8.2 hat laterale Bewegung nach einer beliebigen einzelnen Kompromittierung als größtes Risiko markiert. Eine Umstellung auf einen Schlag würde den Datenverkehr unterbrechen, daher verläuft die Migration in Stufen, und jeder Schritt ist umkehrbar.

1. Inventarisieren und beobachten (Wochen 1 bis 3)	Jeden Service, sein zuständiges Team und seine Aufrufer erfassen. Das Mesh im permissiven Modus installieren, der sowohl einfachen als auch mTLS-Datenverkehr annimmt, und mit seiner Telemetrie den tatsächlichen Aufrufgraphen erstellen. Gefundene Überraschung: Der Analytics-Job greift direkt auf die Datenbank von Zahlungen zu.
2. Jedem Workload eine Identität geben (Wochen 3 bis 5)	Ein Kubernetes-Service-Account pro Service; das Mesh stellt SPIFFE-Zertifikate mit 24 Stunden Gültigkeit aus, die automatisch rotiert werden. Der gemeinsame Datenbanknutzer wird durch einen Berechtigungsnachweis pro Service ersetzt, den der Secrets-Manager jeweils für eine Stunde ausstellt.
3. Alles verschlüsseln (Wochen 5 bis 8)	Namespaces einzeln auf striktes mTLS umstellen, beginnend mit den risikoarmen. Dashboards zeigen alle verbleibenden Klartext-Aufrufer; jeder wird behoben, bevor sein Namespace umgestellt wird. Ein Rollback ist eine einzige Konfigurationsänderung pro Namespace.
4. Jeden Aufruf autorisieren (Wochen 8 bis 12)	Allow-Richtlinien aus dem beobachteten Aufrufgraphen schreiben: Bestellungen darf Zahlungen über `POST /authorisations` aufrufen; Speisekarte darf in Zahlungen nichts aufrufen. Die Richtlinien zwei Wochen im Audit-Modus laufen lassen, die Ablehnungen prüfen, dann durchsetzen. Default-Deny-NetworkPolicies in Kubernetes und Egress-Regeln als äußere Ebene ergänzen.
5. Das VPN für die Belegschaft ersetzen (Wochen 10 bis 16)	Die Admin-Konsole und interne Tools hinter einen Identity-Aware Proxy stellen, der Single Sign-on, Phishing-resistente MFA und ein verwaltetes, gepatchtes Gerät verlangt. Erstattungen über einem Schwellenwert erfordern einen erneuten MFA-Schritt. Das VPN wird abgeschaltet, sobald es nichts mehr nutzt.
6. Den Rand härten (parallel)	Das Gateway validiert Tokens, wendet Rate Limits pro Route und Schemavalidierung an und leitet eine verifizierte Identität weiter; jeder Service prüft weiterhin, ob der Aufrufer die konkrete Bestellung oder das konkrete Konto sehen darf.

7. Messen und dauerhaft einhalten	Fitness Functions lassen den Build scheitern, wenn ein neuer Service keine Autorisierungsrichtlinie hat, und Alarme lösen bei abgelehnten Aufrufen und bei Zertifikaten aus, die in weniger als sieben Tagen ablaufen. Jede neue Abhängigkeit ist eine geprüfte Richtlinienänderung.

Die Reihenfolge zählt: beobachten vor dem Durchsetzen, Identität vor Richtlinie, Audit-Modus vor Ablehnung. Jeder Schritt lässt sich für sich zurücknehmen, wodurch das Risiko der Migration geringer bleibt als das Risiko, das sie beseitigt.

Das Ergebnis ist nicht, dass nichts mehr kompromittiert werden kann. Ein kompromittierter Speisekarten-Service kann seine eigenen Daten und die ihm erlaubten Aufrufe weiterhin missbrauchen, aber er erreicht weder Zahlungen noch die Datenbanken anderer Services, und seine Versuche werden protokolliert und lösen Alarme aus.

ACHTUNG

Verbreitete Irrtümer über Zero Trust

- **Es ist kein Produkt.** Das NIST beschreibt Zero Trust als Prinzipien und als Architektur; Anbieter verkaufen Komponenten (Proxys, Identity Provider, Meshes), die Teile davon umsetzen.
- **Es schafft die Netzwerkkontrollen nicht ab.** Segmentierung und Firewalls bleiben als äußere Ebenen; sie sind nur nicht mehr das Einzige zwischen einem Angreifer und den Daten.
- **mTLS allein ist kein Zero Trust.** Verschlüsselung mit Identität, aber einer Allow-All-Richtlinie lässt jeden kompromittierten Service weiterhin alles aufrufen; um die Autorisierung pro Aufruf geht es.
- **Es ist nicht mit einem Projekt erledigt.** Das Reifegradmodell der CISA gibt es, weil Organisationen über Jahre Säule für Säule vorankommen.
- **Es darf Sicherheit nicht unbenutzbar machen.** Die psychologische Akzeptanz nach Saltzer und Schroeder gilt weiterhin: Häufige Abfragen bringen Menschen bei, alles zu bestätigen, daher sollten zusätzliche Prüfungen riskanten Aktionen vorbehalten bleiben.

Häufige Fragen

Brauchen wir noch ein VPN?
Für Menschen, die auf Webanwendungen zugreifen, ersetzt es meist ein Identity-Aware Proxy. VPNs oder private Verbindungen bleiben sinnvoll für Standortverbindungen und für Protokolle, die ein Proxy nicht verarbeiten kann, sollten aber dem, was auf der anderen Seite ist, kein weitreichendes Vertrauen einräumen.

Ist für Zero Trust ein Service Mesh nötig?
Nein. Es ist ein bequemer Weg, mTLS, Workload Identity und Richtlinien pro Aufruf überall zu erhalten. Load Balancer der Plattform, Bibliotheken und Identitätsfunktionen der Cloud können bei wenigen Services dasselbe leisten.

Macht Zero Trust das System langsamer?
Jede Prüfung fügt etwas Latenz hinzu, typischerweise deutlich unter einer Millisekunde für einen lokalen Mesh-Proxy und einige Millisekunden, wenn eine entfernte Entscheidung nötig ist. Entscheidungen zu cachen, Richtlinien an den Durchsetzungspunkt zu verteilen und Verbindungen wiederzuverwenden hält sie gering; messen Sie sie beim p99 auf kritischen Pfaden.

Browsersicherheit: Same-Origin-Policy, CORS und CSRF

Bisher waren die Aufrufer Services und Apps, die ihre eigenen Zugangsdaten halten. Ein Browser ist anders: Er führt Code vieler Websites nebeneinander aus und hängt die Cookies eines Nutzers automatisch an. Die Kundenwebsite und das Restaurantportal der Plattform haben daher eine Vertrauensgrenze im Browser des Kunden, wo kein Gateway hinkommt. Der Ausgangspunkt ist die **Same-Origin-Policy**. Zwei URLs teilen sich nur dann einen Origin, wenn Schema, Host und Port alle übereinstimmen, und Skripte eines Origins sind eingeschränkt darin, wie sie auf Ressourcen eines anderen zugreifen können. Die Einschränkung trifft vor allem das *Lesen*: Cross-Origin-Lesezugriffe werden typischerweise blockiert, Cross-Origin-Schreibzugriffe wie Formularübermittlungen typischerweise erlaubt.

Cross-Origin Resource Sharing (CORS) ist die kontrollierte Lockerung dieser Regel. Ruft die Website auf `app.example.com` die Adresse `api.example.com` auf, sind das verschiedene Origins, also muss die API mit `Access-Control-Allow-Origin` antworten und die Website nennen. Bei allem, was über eine einfache Anfrage hinausgeht (ein `PUT` oder `DELETE`, ein eigener Header, ein JSON-Body), fragt der Browser zuerst mit einem `OPTIONS`-Preflight um Erlaubnis. Drei Punkte werden oft missverstanden. Der Browser setzt CORS durch, der Server erklärt nur eine Policy; Tools wie `curl` ignorieren sie also, und die API muss weiterhin jeden Aufruf authentifizieren und autorisieren. CORS lockert die Same-Origin-Policy immer nur; es

fügt keinen Schutz hinzu. Und eine einfache Anfrage erreicht den Server trotzdem und kann Wirkung entfalten: Nur die Antwort wird zurückgehalten. Ein Wildcard-Origin wird bei Anfragen mit Zugangsdaten abgelehnt, und jeden eintreffenden `Origin` einfach zurückzugeben, verfehlt den Zweck; vergleichen Sie ihn also mit einer Allowlist.

Cross-Site Request Forgery (CSRF) nutzt genau diese Lücke aus. Eine Seite einer anderen Website veranlasst den Browser, eine Anfrage an die Plattform zu senden, das Cookie des Kunden geht womöglich mit, und der Server sieht etwas, das aussieht, als ändere der Kunde eine Lieferadresse. Die wichtigsten Abwehrmaßnahmen von OWASP sind ein unvorhersagbares Token, das bei jeder zustandsändernden Anfrage geprüft wird (pro Sitzung, wo der Server Zustand hält, ein signiertes Double-Submit-Cookie, wo nicht), oder eine Prüfung des `Sec-Fetch-Site`-Headers mit Rückfall auf die Prüfung von `Origin`. Das `SameSite`-Cookie-Attribut aus Abschnitt 8.3 ist nützliche Defense in Depth, aber kein Ersatz: `Lax` sendet Cookies weiterhin bei seitenübergreifenden `GET`-Navigationen auf oberster Ebene, ein `GET` darf also nie den Zustand ändern. Ein per Skript in einem `Authorization`-Header gesendetes Token wird nicht automatisch angehängt und ist auf diese Weise nicht angreifbar, doch ein Backend for Frontend bringt Cookies zurück und braucht CSRF-Schutz. Eine Warnung: Cross-Site-Scripting hebelt all diese Abwehrmaßnahmen aus, was zur nächsten Kontrolle führt.

Content Security Policy und Sicherheits-Header

Content Security Policy (CSP) ist ein Antwort-Header, der dem Browser sagt, was eine Seite laden und ausführen darf. Ihr Hauptnutzen ist, den Schaden durch Cross-Site-Scripting zu begrenzen: Mit einer Policy werden eingeschleuste Inline-Skripte, Inline-Event-Handler und `eval()` blockiert, sofern die Policy sie nicht ausdrücklich erlaubt. Die W3C-Spezifikation macht deutlich, dass CSP am besten als Defense in Depth genutzt wird: Sie verringert den Schaden einer Injection, ersetzt aber nicht die Eingabevalidierung und die Ausgabekodierung. CSP Level 3 ist noch ein Working Draft, prüfen Sie also die aktuelle Version und die Browserunterstützung.

Allowlists von Hosts lassen sich oft umgehen und sind schwer zu pflegen, deshalb bevorzugen aktuelle Empfehlungen eine **Strict CSP** auf Basis von Nonces oder Hashes, zum Beispiel `script-src 'nonce-RANDOM'; object-src 'none'; base-uri 'none'`. Eine Nonce muss bei jeder Antwort ein neuer, unvorhersagbarer Wert sein, was eine architektonische Folge hat: Seiten müssen pro Antwort erzeugt werden, während zu gecachten statischen Seiten Hashes passen, die bei jeder Skriptänderung neu berechnet werden müssen. Jedes Drittanbieter-Skript, etwa ein Zahlungs-Widget oder ein Analyse-Tag, muss inventarisiert und erlaubt werden. Der sichere Rollout ist `Content-Security-Policy-Report-Only`: Die Policy meldet Verstöße an einen Endpoint, ohne etwas zu blockieren, und wird durchgesetzt, sobald die Berichte sauber sind. Die Direktive `frame-ancestors 'none'` verhindert zudem, dass andere Websites die Checkout-Seite in Frames einbetten, eine Abwehr gegen Clickjacking.

Einige weitere Header sind günstig und überall sinnvoll. `Strict-Transport-Security` (HSTS, RFC 6797) weist den Browser an, für die Website nur HTTPS zu verwenden; weil der allererste Besuch ungeschützt ist, können Websites auch die Aufnahme in die Preload-Liste der Browser beantragen. `X-Content-Type-Options: nosniff` hindert den Browser daran, Inhaltstypen zu erraten. `Referrer-Policy: strict-origin-when-cross-origin` hält Pfade und Bestellreferenzen aus dem Referrer heraus, der an andere Websites gesendet wird. `Permissions-Policy` schaltet Browserfunktionen ab, die eine Seite nicht braucht; die Kurierseite braucht womöglich Geolocation, die Speisekartenseite nicht. OWASP rät davon ab, sich auf den alten `X-XSS-Protection`-Header zu verlassen, und empfiehlt, ihn auf `0` zu setzen; prüfen Sie das Cheat Sheet auf aktuelle Werte.

Für den Architekten ist die Frage, wo diese Einstellungen liegen. Setzt jedes Team sie von Hand, wird ein Service sie vergessen. Setzen Sie sie einmal an der Edge oder im Gateway oder in einer gemeinsamen Vorlage, und testen Sie die deployten Antworten in der Pipeline als Fitness Function (Abschnitt 11.3).

PRÜFEN SIE IHR VERSTÄNDNIS

▼ Warum reicht Mutual TLS zwischen Services, bei dem jeder Service jeden anderen aufrufen darf, für Zero Trust nicht aus?

Es bietet Verschlüsselung und Identität, aber keine Autorisierung. Ein kompromittierter Service kann weiterhin alles aufrufen; Zero Trust braucht eine Richtlinie pro Aufruf, die nur die Aufrufe erlaubt, die jede Identität tatsächlich benötigt.

▼ Was ist der Unterschied zwischen der Policy Engine und dem Policy Enforcement Point?

Die Policy Engine entscheidet anhand von Richtlinie und Eingaben wie Identität und Sicherheitszustand, ob eine Anfrage erlaubt werden soll; der Enforcement Point sitzt im Datenpfad und setzt diese Entscheidung um, indem er die Anfrage zulässt oder blockiert.

▼ Warum sind kurzlebige Zertifikate attraktiv, und was setzen sie voraus?

Ein gestohlener Schlüssel ist nur nützlich, bis das Zertifikat abläuft, was die Abhängigkeit vom Widerruf verringert. Sie setzen eine vollständig automatisierte Ausstellung und Erneuerung voraus, dazu eine Überwachung der Ablaufdaten.

▼ Ein Kunde ändert die Bestellnummer in einer URL und sieht die Bestellung eines anderen Kunden. Welche Komponente hätte das verhindern sollen?

Der Service, dem die Bestellungen gehören, indem er prüft, ob dem authentifizierten Kunden genau diese Bestellung gehört. Das Gateway hat den Nutzer korrekt authentifiziert, kann aber nicht wissen, wem jedes einzelne Objekt gehört.

▼ **Probieren Sie es aus: Ein Team sagt: „Das API-Gateway validiert jedes Token, und unsere Services liegen in einem privaten Netzwerk, also brauchen die Services keine weiteren Prüfungen.“ Was könnte schiefgehen, und was würden Sie nach dem Zero-Trust-Denken ergänzen?**

Ein kompromittierter Service oder gestohlene Zugangsdaten können sich lateral bewegen und alles aufrufen, weil der Standort als Vertrauen gilt. Ein gültiges Token zeigt auch nicht, dass dieser Kunde diese Bestellung sehen darf. Geben Sie jedem Service eine verifizierte Identität mit Mutual TLS, idealerweise kurzlebig und automatisch ausgestellt, und erlauben Sie nur die nötigen Aufrufe. Jeder Service muss die Prüfung auf Objektebene für Daten durchführen, die ihm gehören. Das Gateway bleibt für Tokens, Rate Limits und Eingabevalidierung nützlich.

QUELLEN

- NIST SP 800-207: Zero Trust Architecture (2020): https://csrc.nist.gov/pubs/sp/800/207/final
- Ward and Beyer: BeyondCorp, A New Approach to Enterprise Security (2014): https://research.google/pubs/beyondcorp-a-new-approach-to-enterprise-security/
- RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3: https://www.rfc-editor.org/rfc/rfc8446
- RFC 8555: Automatic Certificate Management Environment (ACME): https://www.rfc-editor.org/rfc/rfc8555
- SPIFFE: overview of the SPIFFE standards: https://spiffe.io/docs/latest/spiffe-about/overview/
- MDN: Same-origin policy: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy
- MDN: Cross-Origin Resource Sharing (CORS): https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CORS
- OWASP Cross-Site Request Forgery Prevention Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html
- MDN: Content Security Policy (CSP): https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CSP
- W3C: Content Security Policy Level 3: https://www.w3.org/TR/CSP3/
- OWASP HTTP Headers Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Headers_Cheat_Sheet.html
- RFC 6797: HTTP Strict Transport Security (HSTS): https://www.rfc-editor.org/rfc/rfc6797.html

Sicherheit der Software-Lieferkette

BEVOR SIE DAS LESEN

Dieser Abschnitt baut auf:

- 0.3 Code organisieren
- 0.8 Wie Software entsteht
- 8.1 Sicherheitsprinzipien

Begriffe von früher, die er verwendet: Abhängigkeit, CI/CD-Pipeline, Container-Image, Angriffsfläche, Least Privilege

DIE IDEE IN KÜRZE

Ein Restaurant baut weder sein Mehl selbst an noch baut es seine Öfen selbst; es vertraut Lieferanten. Ist das Mehl eines Lieferanten verunreinigt, ist jedes Restaurant betroffen, das es gekauft hat, wie sauber seine eigene Küche auch sein mag. Gute Restaurants führen eine Liste jeder Zutat und ihrer Herkunft, kaufen bei Lieferanten, die ihre Lieferungen versiegeln und kennzeichnen, und prüfen die Siegel bei der Ankunft. Software entsteht auf dieselbe Weise, aus Hunderten von Komponenten, die andere geschrieben haben und die Build-Maschinen zusammensetzen. Sicherheit der Software-Lieferkette heißt zu wissen, was in Ihrer Software steckt, woher es kommt und dass niemand es unterwegs manipuliert hat.

WAS SIE WISSEN MÜSSEN

- Der meiste Code in einem modernen Service stammt aus Abhängigkeiten, Build-Tools, CI-Actions und Basis-Images; jedes davon und jeder, der es ändern kann, gehört zur Angriffsfläche.
- event-stream (2018) missbrauchte das Vertrauen in Maintainer, SolarWinds (2020) ein kompromittiertes Build-System, Log4Shell (2021) eine verwundbare Bibliothek tief in Abhängigkeitsbäumen und xz utils (2024) die geduldige Übernahme eines Release-Prozesses.
- Eine SBOM im Format SPDX oder CycloneDX listet Komponenten mit exakten Kennungen auf, sodass sich die Betroffenheit von einer neuen Schwachstelle per Abfrage ermitteln lässt; die US Executive Order 14028 (2021) und die Cyberresilienz-Verordnung der EU haben SBOMs zum Standard gemacht.

- SLSA (Version 1.2) definiert die Build-Level L0 bis L3, bis hin zu gehärteten, isolierten Builds, sowie Source-Level bis zum Review durch zwei Personen; Provenienz hält fest, wer ein Artefakt woraus gebaut hat.
- Das schlüssellose Signieren von Sigstore bindet ein kurzlebiges Zertifikat an eine CI-Identität und erfasst Signaturen in einem öffentlichen Transparenzlog; Admission-Prüfungen verifizieren Signaturen und Provenienz vor dem Deployment.
- Lockfiles mit Hashes, das Pinnen von CI-Actions und Images per Digest, kleine automatisierte Updates mit kurzer Wartezeit für neue Releases, priorisiertes Scannen sowie ephemere, hermetische Builds verringern das Risiko im Arbeitsalltag.
- Die Meldepflichten der Cyberresilienz-Verordnung gelten ab dem 11. September 2026 (Frühwarnung binnen 24 Stunden, Meldung binnen 72 Stunden), die meisten übrigen Pflichten ab dem 11. Dezember 2027.

Warum Abhängigkeiten eine Angriffsfläche sind

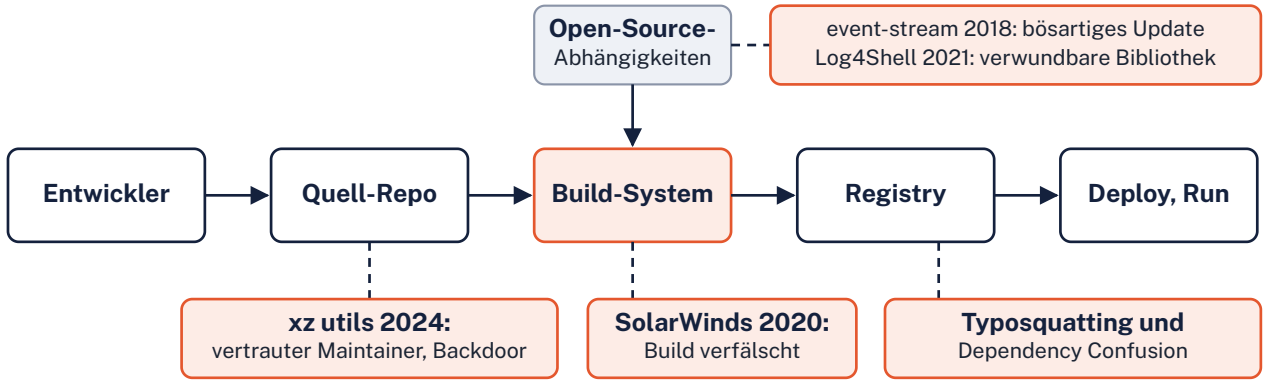
Der meiste Code in einem modernen Service wurde nicht von dem Team geschrieben, das ihn betreibt. Ein kleiner Webservice deklariert vielleicht zwanzig direkte Abhängigkeiten und installiert über deren eigene Abhängigkeiten mehrere hundert Pakete; der Container, in dem er ausgeliefert wird, bringt ein Betriebssystem-Userland mit, und die Pipeline, die ihn baut, führt Compiler, Plug-ins und CI-Actions aus, die wiederum andere geschrieben haben. Abschnitt 0.3 stellte Abhängigkeiten als Aufwandsersparnis mit Wartungskosten vor. Dieser Abschnitt betrachtet ihre Sicherheitskosten: Jede dieser Komponenten und jede Person und jedes System, die sie ändern können, gehört zu Ihrer Angriffsfläche.

Ein Software-Supply-Chain-Angriff kompromittiert etwas, das Sie nutzen, um Sie zu erreichen, meist viele Opfer auf einmal. Er kann auf vier grundlegenden Wegen kommen. **Schadcode aus einer vertrauenswürdigen Quelle:** Das Konto eines Maintainers wird per Phishing übernommen, ein Maintainer wird feindselig, oder ein Projekt wird einem Fremden übergeben. **Bösartige Doppelgänger:** Typosquatting veröffentlicht `reqeusts` neben `requests`; *Dependency Confusion*, im Februar 2021 von Alex Birsan demonstriert, veröffentlicht in der öffentlichen Registry ein Paket mit demselben Namen wie ein internes Paket eines Unternehmens und einer höheren Versionsnummer, das manche Build-Tools dann bevorzugen. **Manipulation in der Pipeline:** Der Quellcode ist in Ordnung, aber das Build-System, die Signaturschlüssel oder die Registry sind kompromittiert. **Ehrliche Schwachstellen:** Am Schreiben des Codes war kein Angreifer beteiligt, aber ein schwerer Fehler in einer weit verbreiteten transitiven Abhängigkeit gefährdet jeden, der sie ausliefert.

Die letzten Jahre haben gezeigt, wie schnell sich die erste Art ausbreitet. Im März 2025 wurden die Versions-Tags einer beliebten GitHub Action, tj-actions/changed-files, auf einen bösartigen Commit verschoben, der CI-Secrets in Build-Logs ausgab; betroffen waren Tausende

Repositorys, die per Tag auf die Action verwiesen. Im September 2025 wurde ein Maintainer weit verbreiteter npm-Pakete wie `chalk` und `debug` über eine gefälschte npm-Support-Seite per Phishing getäuscht, und bösartige Versionen wurden veröffentlicht und binnen Stunden wieder entfernt; später im selben Monat stahl ein sich selbst verbreitender Wurm mit dem Spitznamen Shai-Hulud npm-Tokens und veröffentlichte damit infizierte Versionen weiterer Pakete.

Die Lieferkette und die Vertrauensbeziehungen, die Angreifer missbrauchen



Jeder Pfeil ist eine Vertrauensbeziehung; jeder bekannte Vorfall missbrauchte eine andere

Quellcode, Build, Registry und Abhängigkeiten sind getrennte Vertrauensbeziehungen, und jeder bekannte Vorfall griff eine andere an.

Vier Vorfälle, genau erzählt

event-stream (2018). event-stream war ein beliebtes npm-Paket mit rund zwei Millionen Downloads pro Woche, für das sein ursprünglicher Autor keine Zeit mehr hatte. Im September 2018 übergab er die Veröffentlichungsrechte an einen Freiwilligen, der Hilfe angeboten hatte. Der neue Maintainer fügte eine Abhängigkeit hinzu, `flatmap-stream`, deren veröffentlichtes Paket eine verschlüsselte Nutzlast enthielt, die nur im Build von Copay, einer Bitcoin-Wallet-Anwendung, aktiv wurde und versuchte, Wallets mit hohen Guthaben zu stehlen. Ein Entwickler bemerkte den verdächtigen Code im November 2018. Nichts am Modell von npm war gebrochen worden: Ein vertrauenswürdiger Name hatte einfach den Besitzer gewechselt.

SolarWinds (2020). Angreifer, die die US-Regierung später dem russischen Auslandsgeheimdienst zuordnete, verschafften sich Zugang zur Build-Umgebung von SolarWinds, dem Hersteller des Netzwerkmanagement-Produkts Orion. Ein Implantat (von den Ermittlern SUNSPOT genannt) wartete auf Orion-Builds, tauschte während der Kompilierung eine Quelldatei gegen eine Version mit Backdoor aus und legte anschließend das Original zurück. Die so entstandenen Updates, veröffentlicht zwischen März und Juni 2020, wurden von SolarWinds' eigener Pipeline gebaut und mit dem echten Zertifikat des Unternehmens signiert. Rund 18.000 Kunden installierten sie; die Angreifer wählten dann eine viel kleinere Zahl von

Behörden und Unternehmen für weitere Eindringversuche aus. FireEye entdeckte die Kampagne im Dezember 2020 bei der Untersuchung eines Angriffs auf das eigene Unternehmen. Die Lehre für Architekten: Eine Signatur beweist, wer etwas gebaut hat, nicht, dass der Build sauber war.

Log4Shell (2021). CVE-2021-44228, am 9. Dezember 2021 offengelegt und nach CVSS mit 10,0 bewertet, war ein Fehler in Log4j 2, der dominierenden Logging-Bibliothek für Java. Log4j löste Lookups auf, die in Log-Nachrichten standen, darunter JNDI-Lookups, die eine Klasse von einem entfernten Server laden konnten; ein Angreifer, der eine Zeichenkette wie einen Benutzernamen oder einen Request-Header ins Log bringen konnte, konnte so Code auf dem Server ausführen. Kein Angreifer hatte die Lieferkette angetastet; der Schaden entstand dadurch, wie weit verbreitet und wie tief eingebettet die Bibliothek war. Organisationen verbrachten Wochen damit herauszufinden, wo sie sie nutzten, mehrere Folge-Releases waren nötig, um verwandte Fehler zu schließen, und wer ein genaues Inventar hatte, beantwortete die Frage in Stunden.

xz utils (2024). xz ist eine Kompressionsbibliothek, die über andere Bibliotheken vom OpenSSH-Server mehrerer Linux-Distributionen genutzt wird. Über etwa zwei Jahre leistete ein Konto namens Jia Tan nützliche Beiträge und wurde, unterstützt von weiteren Konten, die den überlasteten ursprünglichen Maintainer drängten, Hilfe anzunehmen, zum Co-Maintainer. Die Versionen 5.6.0 und 5.6.1, veröffentlicht im Februar und März 2024, enthielten eine Backdoor, mit der der Inhaber eines bestimmten privaten Schlüssels Befehle über SSH hätte ausführen können. Der Schadcode war in binären Testdateien versteckt und wurde durch ein verändertes Build-Skript aktiviert, das nur in den Release-Tarballs existierte, nicht im öffentlichen Repository. Am 29. März 2024 machte Andres Freund, ein Entwickler bei Microsoft, sie öffentlich, nachdem er untersucht hatte, warum SSH-Logins auf einem Testsystem ungewöhnlich viel CPU verbrauchten und etwa eine halbe Sekunde länger dauerten; sie wurde als CVE-2024-3094 erfasst. Weil sie früh entdeckt wurde, hatte sie vor allem Entwicklungs- und Rolling-Release-Distributionen erreicht und kaum stabile.

Was jeder Vorfall lehrt

Vorfall	Was geschah	Angegriffenes Glied	Kontrollen, die geholfen hätten
event-stream (npm), 2018	Ein neuer, freiwilliger Maintainer, dem der erschöpfte ursprüngliche Autor das Paket übergeben hatte, fügte eine Abhängigkeit (flatmap-stream) hinzu, die eine verschlüsselte Nutzlast gegen eine bestimmte Bitcoin-Wallet-Anwendung enthielt.	Vertrauen in Maintainer und transitive Abhängigkeiten	Lockfiles und eine Wartezeit vor der Übernahme neuer Versionen; Prüfung neuer transitiver Abhängigkeiten; Finanzierung und Nachfolgeregelung für kritische Pakete.
SolarWinds Orion, 2020	Angreifer im Netzwerk des Herstellers platzierten Code, der während des Builds eine Quelldatei austauschte, sodass rechtmäßig signierte Updates die Backdoor SUNBURST enthielten.	Build-System	Isolierte, ephemere, überwachte Builds; Provenienz; reproduzierbare Builds zum Vergleich der Ergebnisse; minimale Rechte für das Netzwerkmanagement-Tool selbst.
Log4Shell (Log4j 2), 2021	Eine seit Langem vorhandene Funktion, die JNDI-Lookups in geloggtem Text auflöste, ließ jeden Angreifer, der eine Zeichenkette ins Log bringen konnte, Code auf dem Server ausführen.	Eine verwundbare, tief verschachtelte Abhängigkeit	Eine SBOM, um jedes betroffene System in Minuten zu finden; Egress-Filterung; schnelles, eingeübtes Patchen.
xz utils, 2024	Nach etwa zwei Jahren geduldiger Beiträge lieferte ein Co-Maintainer Release-Tarballs mit einer versteckten Backdoor aus, die auf OpenSSH in einigen Linux-Distributionen zielte.	Quellcode und Release-Prozess	Builds aus dem geprüften Repository statt aus handgefertigten Tarballs; reproduzierbare Builds; genaue Prüfung binärer Testdateien und Build-Skripte.

Software Bills of Materials

Eine Software Bill of Materials (SBOM) ist eine maschinenlesbare Liste der Komponenten einer Software mit Versionen, Kennungen und den Beziehungen zwischen ihnen. Sie macht Software nicht von selbst sicherer; sie macht Fragen beantwortbar. Wird das nächste Log4Shell bekannt, verwandelt ein durchsuchbarer Bestand an SBOMs für jedes deployte Artefakt die Frage „Sind wir betroffen, und wo?“ in eine Abfrage.

Zwei Formate dominieren. SPDX von der Linux Foundation begann als Format für Lizenz-Compliance; Version 2.2.1 ist die internationale Norm ISO/IEC 5962:2021, und SPDX 3.0 (2024) ergänzt Profile für Sicherheit, Builds und KI. CycloneDX, ein OWASP-Projekt für Sicherheitsanwendungsfälle, ist von Ecma International als ECMA-424 standardisiert; Version 1.7 erschien im Oktober 2025. Beide können Komponenten mit einer Package URL (purl) wie `pkg:npm/express@4.21.2` identifizieren, und genau damit gleichen Schwachstellendatenbanken ab. Die ergänzende Idee VEX (Vulnerability Exploitability eXchange) erlaubt es einem Lieferanten zu erklären, dass eine gelistete verwundbare Komponente in seinem Produkt nicht ausnutzbar ist, etwa weil die verwundbare Funktion nie aufgerufen wird; das verringert das Rauschen für die Kunden.

Die Politik hat SBOMs in den Mainstream gebracht. Die US Executive Order 14028, *Improving the Nation's Cybersecurity*, unterzeichnet am 12. Mai 2021, legte Maßnahmen für die Software-Lieferkette bei Software fest, die an die Bundesregierung verkauft wird, darunter die Bereitstellung einer SBOM für Käufer; die National Telecommunications and Information Administration veröffentlichte daraufhin die Mindestelemente einer SBOM (Lieferant, Komponentenname, Version, weitere eindeutige Kennungen, Abhängigkeitsbeziehungen, Autor der SBOM-Daten und ein Zeitstempel). Die US-Politik hat sich seither von einheitlichen Vorgaben entfernt: Im Januar 2026 hob das Memorandum M-26-05 des Office of Management and Budget die frühere Pflicht von Lieferanten auf, sichere Entwicklungspraktiken zu bescheinigen; die Behörden legen nun risikobasierte Anforderungen fest und verlangen SBOMs, wo sie es für richtig halten. In der EU macht die weiter unten beschriebene Cyberresilienz-Verordnung eine SBOM zum Teil der technischen Dokumentation für Produkte mit digitalen Elementen.

Erzeugen Sie SBOMs während des Builds, aus dem, was tatsächlich aufgelöst und paketiert wurde, statt aus dem Manifest, das ein Entwickler geschrieben hat; speichern Sie sie neben dem Artefakt, mit dessen Digest als Schlüssel; und speisen Sie sie in einen Service ein, der jede gespeicherte SBOM erneut prüft, sobald eine neue Schwachstelle veröffentlicht wird.

```
{
  "bomFormat": "CycloneDX",
  "specVersion": "1.7",
  "metadata": {"timestamp": "2026-10-20T09:14:02Z",
               "component": {"bom-ref": "orders-service", "type": "application",
                             "name": "orders-service", "version": "4.12.0"}},
  "components": [
    {"bom-ref": "log4j-core", "type": "library", "name": "log4j-core",
     "group": "org.apache.logging.log4j", "version": "2.24.3",
     "purl": "pkg:maven/org.apache.logging.log4j/log4j-core@2.24.3",
     "hashes": [{"alg": "SHA-256", "content": "9f2c..."}]},
    {"bom-ref": "jackson-databind", "type": "library", "name": "jackson-databind",
     "version": "2.18.2",
     "purl": "pkg:maven/com.fasterxml.jackson.core/jackson-databind@2.18.2"}
  ],
  "dependencies": [
    {"ref": "orders-service", "dependsOn": ["log4j-core", "jackson-databind"]}
  ]
}
```

Eine gekürzte CycloneDX-SBOM für den Bestellservice der Essenslieferplattform. purl und Hash identifizieren jede Komponente exakt, sodass ein Scanner sie mit Schwachstellendatenbanken abgleichen kann.

SLSA: Level für die Build-Integrität

SLSA (Supply-chain Levels for Software Artifacts, ausgesprochen *salsa*) ist ein Framework, inzwischen unter dem Dach der Open Source Security Foundation gepflegt, das aufeinander aufbauende Schutzstufen gegen Manipulation beschreibt. Version 1.0 (2023) definierte einen Build-Track; Version 1.2 gliederte die Spezifikation in mehrere Tracks und ergänzte einen Source-Track. Das zentrale Artefakt ist die Provenienz: eine signierte Aussage, meist im Attestierungsformat von in-toto, darüber, welche Build-Plattform ein Artefakt erzeugt hat, aus welcher Quellcode-Revision und mit welcher Build-Definition und welchen Parametern. Ein Konsument prüft die Provenienz gegen Erwartungen wie *von unserer CI aus dem main-Branch von github.com/fooddelivery/orders gebaut*.

Der Source-Track hat vier Level: Versionskontrolle (L1), eine fortlaufende, unveränderliche Historie mit Provenienz für jede Revision (L2), fortlaufend durchgesetzte technische Kontrollen wie geschützte Branches (L3) und ein Review durch zwei Personen für jede Änderung an geschützten Branches (L4). Für die meisten Organisationen ist das realistische Ziel Build L3 auf einer gemeinsamen, gut betriebenen Build-Plattform plus Source L3 oder L4 für wichtige Repositorys; die Level sagen, was eine Plattform garantieren muss, und ein Plattformteam kann sie einmal für alle Services erfüllen.

Die Level des SLSA-Build-Tracks

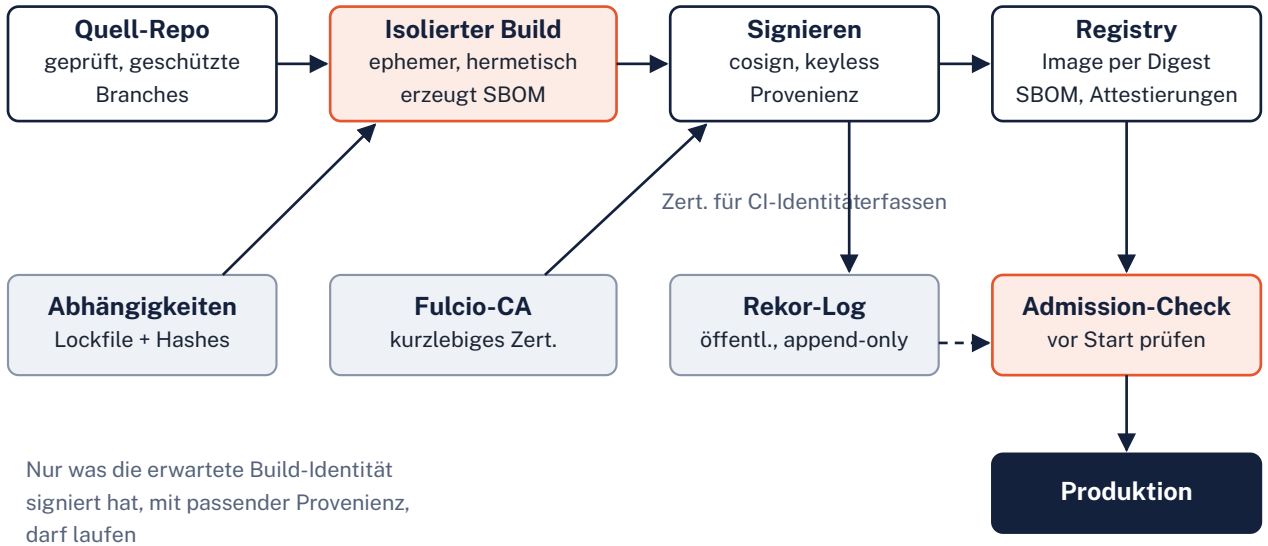
Niveau	Name	Was es garantiert
Build L0	Keine Garantien	Keine Anforderungen: das Fehlen von SLSA.
Build L1	Provenienz vorhanden	Der Build erzeugt Provenienz, die beschreibt, wie das Artefakt gebaut wurde. Das fängt Fehler ab, ist aber leicht zu fälschen.
Build L2	Gehostete Build-Plattform	Ein gehosteter Build-Dienst erzeugt und signiert die Provenienz und schützt so vor Manipulation nach dem Build.
Build L3	Gehärtete Builds	Die Build-Plattform isoliert Builds voneinander und hält Signiermaterial außer Reichweite benutzerdefinierter Build-Schritte; das schützt vor Manipulation während des Builds.

Signieren und Provenienz mit Sigstore

Artefakte zu signieren ist seit Langem möglich, doch die Verwaltung langlebiger Signaturschlüssel war so mühsam, dass nur wenige Open-Source-Projekte es taten, und gestohlene Schlüssel waren selbst ein Risiko. Sigstore, 2021 gestartet und heute Teil der Open Source Security Foundation, macht das Signieren durch *schlüsselloses* Signieren billig. Ein CI-Job weist Fulcio, der Zertifizierungsstelle von Sigstore, seine Identität mit einem OpenID-Connect-Token nach; Fulcio stellt ein wenige Minuten gültiges Zertifikat aus, das diese Identität nennt, etwa einen bestimmten Workflow in einem bestimmten Repository; das Tool signiert das Artefakt und erfasst die Signatur in Rekor, einem öffentlichen Transparenzlog, an das nur angefügt werden kann; der ephemere private Schlüssel wird danach verworfen. Prüfer kontrollieren die Signatur, die Identität im Zertifikat und den Log-Eintrag, statt einem Schlüssel zu vertrauen, der durchgesickert sein könnte.

Das Tool cosign wendet das auf Container-Images und andere Artefakte an, und Paket-Ökosysteme haben dieselben Ideen übernommen: npm kann Pakete mit Provenienzangaben veröffentlichen, die sie mit dem Quell-Repository und dem Build verknüpfen (2023), und PyPI unterstützt Trusted Publishing aus der CI ohne langlebige Tokens (2023) sowie auf Sigstore basierende Attestierungen (2024). Beim Deployment kann ein Admission Controller in Kubernetes, etwa der Policy Controller von Sigstore oder Kyverno, jedes Image ablehnen, das nicht von der erwarteten Identität signiert ist oder dem die erwartete Provenienz fehlt. Denken Sie an SolarWinds: Signatur und Provenienz beweisen, woher etwas stammt; erst Build-Isolation, Review und Erkennung machen vertrauenswürdig, was dabei herauskommt.

Eine gehärtete Pipeline vom Commit bis zur Produktion



Jede Stufe erzeugt Nachweise, die die nächste prüfen kann, sodass die Produktion nur ausführt, was die erwartete Pipeline aus geprüftem Quellcode gebaut hat.

Lockfiles, Pinning und automatisierte Updates

Ein Manifest wie `package.json` oder `pyproject.toml` gibt zulässige Versionsbereiche an; ein Lockfile (`package-lock.json`, `poetry.lock`, `uv.lock`, `go.sum`, `Cargo.lock`) hält die exakt aufgelösten Versionen fest, meist mit einem kryptografischen Hash für jede. Wird das Lockfile committet und in der CI daraus installiert (`npm ci`, `pip install --require-hashes`), nutzt jeder Build dieselben geprüften Bytes, und ein mit anderem Inhalt neu veröffentlichtes Paket fällt bei der Hash-Prüfung durch. Pinning gilt auch für Build-Werkzeuge: Verweisen Sie auf CI-Actions per vollständigem Commit-Hash, nicht per Tag, den der Eigentümer oder ein Angreifer verschieben kann, denn genau das ließ den tj-actions-Vorfall sich ausbreiten; verweisen Sie auf Basis-Images per Digest.

Pinning ohne Updates schafft das umgekehrte Risiko: Alte Versionen sammeln bekannte Schwachstellen an, und Upgrades werden zu groß, um sie noch anzugehen. Update-Bots wie Dependabot und Renovate öffnen kleine, häufige Pull Requests, die die CI automatisch testet. Eine nützliche Verfeinerung ist ein Mindestalter für Releases: einige Tage warten, bevor man eine routinemäßige neue Version übernimmt, denn die meisten bösartigen Releases werden innerhalb von Stunden oder Tagen entdeckt und zurückgezogen; Sicherheitskorrekturen sind davon ausgenommen und werden beschleunigt.

Werkzeuge für Software Composition Analysis (SCA) wie OWASP Dependency-Check, Trivy, Grype oder kommerzielle Dienste gleichen Ihre Abhängigkeiten mit Schwachstellendatenbanken ab. Ihre Schwäche ist das Rauschen: Hunderte Befunde, die

meisten in Codepfaden, die Sie nie aufrufen. Priorisieren Sie mit Exploit-Daten (steht die Schwachstelle im Katalog Known Exploited Vulnerabilities der CISA, wie hoch ist ihr EPSS-Wert?), mit Erreichbarkeitsanalyse, wo Ihre Werkzeuge sie bieten, und mit der Exposition (ist der Service aus dem Internet erreichbar?). Abschnitt 8.8 behandelt Schwachstellenmanagement als Prozess.

WENN DAS NEU FÜR SIE IST

Schritt für Schritt: gleicher Code, anderer Build

Der Punkt, an dem hier die meisten stolpern, ist, dass derselbe Quellcode an verschiedenen Tagen unterschiedliche Software ergeben kann, solange die exakten Abhängigkeiten nicht festgeschrieben sind. Verfolgen Sie eine Bibliothek durch zwei Build-Tage des Bestellservice.

- Das Manifest des Bestellservice verlangt eine Bibliothek in „Version 2.18 oder einer späteren 2.x“. Am Montag holt der Build die neueste passende Version, 2.18.2, und die Tests bestehen.
- Am Dienstag erscheint Version 2.18.3, veröffentlicht vom Maintainer oder vielleicht von einem Angreifer, der das Konto des Maintainers per Phishing übernommen hat. Niemand im Team ändert eine Zeile, und doch holt der Build am Dienstag 2.18.3, weil auch diese Version zum Bereich passt.
- Dasselbe kann in den eigenen transitiven Abhängigkeiten der Bibliothek passieren, die das Team nie ausgewählt hat.
- Ein Lockfile verhindert das. Am Montag hält der Paketmanager die exakte Version jedes aufgelösten Pakets fest, transitive eingeschlossen, meist mit einem kryptografischen Hash: einem Fingerabdruck des exakten Paketinhalts. Das Team committet die Datei, und die CI installiert nur, was darin steht.
- Wird 2.18.2 jemals mit anderem Inhalt neu veröffentlicht, passt der Fingerabdruck nicht mehr, und der Build scheitert. Build-Werkzeuge brauchen dieselbe Sorgfalt: Eine per Tag benannte CI-Action kann auf anderen Code verschoben werden, wie beim tj-actions-Vorfall, während ein vollständiger Commit-Hash immer denselben Code bezeichnet.
- Mit dem Festschreiben ist es nicht getan. Alte Versionen sammeln bekannte Schwachstellen an, daher schlägt ein Update-Bot kleine Upgrades vor, die die CI testet. Wer einige Tage wartet, bevor er routinemäßige Releases übernimmt, lässt die meisten bösartigen zuerst auffliegen; Sicherheitskorrekturen gehen schnell durch.
- **Die Lehre:** Entscheiden Sie bewusst, wann sich Ihre Abhängigkeiten ändern. Schreiben Sie exakte Versionen fest, damit jeder Build dieselben geprüften Bytes nutzt, und aktualisieren Sie dann in kleinen, getesteten Schritten.

Container und Build-Integrität

Ein Container-Image enthält neben Ihrem Code ein Betriebssystem-Userland, und jedes Paket darin gehört zur Lieferkette. Beginnen Sie mit minimalen Basis-Images (distroless, Alpine oder schlanke Distributions-Images), die weniger potenziell verwundbare Pakete enthalten; bauen Sie Images regelmäßig neu, damit Korrekturen im Basis-Image ankommen, auch wenn sich Ihr Code nicht geändert hat; scannen Sie Images in der CI und laufend erneut in der Registry, weil jeden Tag neue Schwachstellen für alte Images veröffentlicht werden; und deployen Sie Images per Digest, damit genau das läuft, was gescannt wurde.

Das Build-System ist das wertvollste Ziel in der Kette, weil es alles berührt, wie SolarWinds gezeigt hat. Lassen Sie Builds auf ephemeren Runnern laufen, die für einen einzigen Job erstellt und danach zerstört werden, damit sich ein Angreifer nicht zwischen Builds festsetzen kann. Machen Sie Builds hermetisch, mit allen Eingaben deklariert und per Hash abgerufen und ohne Netzwerkzugriff während der Kompilierung, damit keine nicht deklarierten Eingaben hineingelangen. Halten Sie langlebige Secrets aus der CI heraus, indem Sie per OIDC-Föderation für jeden Job kurzlebige Cloud-Zugangsdaten beziehen, und geben Sie jeder Pipeline nur die geringsten nötigen Rechte, damit ein Pull Request aus einem Fork die Deployment-Zugangsdaten nicht erreichen kann. Schützen Sie den Release-Pfad mit Branch-Schutz und Review durch zwei Personen. Schließlich erlauben reproduzierbare Builds, bei denen derselbe Quellcode stets eine bitgenau identische Ausgabe ergibt, unabhängigen Parteien, neu zu bauen und zu vergleichen; das Projekt Reproducible Builds und Distributionen wie Debian haben die meisten ihrer Pakete reproduzierbar gemacht, und eine Abweichung ist ein starkes Indiz für Manipulation.

Die Cyberresilienz-Verordnung der EU

Die Cyberresilienz-Verordnung (CRA), Verordnung (EU) 2024/2847, legt horizontale Cybersicherheitsanforderungen für *Produkte mit digitalen Elementen* fest, die auf dem Unionsmarkt bereitgestellt werden: Hardware- und Softwareprodukte einschließlich ihrer Lösungen zur Datenfernverarbeitung. Sie ist am 10. Dezember 2024 in Kraft getreten und gilt stufenweise. Die Vorschriften über die Notifizierung von Konformitätsbewertungsstellen gelten ab dem 11. Juni 2026. Die Meldepflichten nach Artikel 14 gelten ab dem 11. September 2026: Ein Hersteller muss eine aktiv ausgenutzte Schwachstelle oder einen schwerwiegenden Sicherheitsvorfall, der die Sicherheit seines Produkts beeinträchtigt, mit einer Frühwarnung innerhalb von 24 Stunden, nachdem er davon Kenntnis erlangt hat, einer ausführlicheren Meldung innerhalb von 72 Stunden und später einem Abschlussbericht melden (bei einer Schwachstelle spätestens 14 Tage, nachdem eine Korrekturmaßnahme verfügbar ist; bei einem schwerwiegenden Vorfall innerhalb eines Monats), und zwar über die einheitliche Meldeplattform, die von ENISA, der Agentur der Europäischen Union für Cybersicherheit, betrieben wird. Die meisten übrigen Pflichten gelten ab dem 11. Dezember 2027.

Die grundlegenden Anforderungen in Anhang I sind ebenso architektonisch wie verfahrensbezogen: Produkte müssen so konzipiert sein, dass sie ein angemessenes Cybersicherheitsniveau gewährleisten, mit einer sicheren Standardkonfiguration bereitgestellt werden, Angriffsflächen begrenzen, die Vertraulichkeit und Integrität von Daten schützen, Daten minimieren sowie sicherheitsrelevante Ereignisse protokollieren und Sicherheitsupdates empfangen können. Hersteller müssen Komponenten ermitteln und dokumentieren, unter anderem durch die Erstellung einer SBOM, die zumindest die obersten Abhängigkeiten erfasst, Schwachstellen über eine Strategie zur koordinierten Offenlegung behandeln und Sicherheitsupdates für einen Unterstützungszeitraum bereitstellen, der in der Regel mindestens fünf Jahre betragen sollte, es sei denn, das Produkt wird voraussichtlich kürzer verwendet. Geldbußen für Verstöße gegen die grundlegenden Anforderungen oder die Pflichten der Hersteller können bis zu 15 Millionen EUR oder 2,5 % des weltweiten Jahresumsatzes betragen, je nachdem, welcher Betrag höher ist.

Der Anwendungsbereich ist entscheidend. Freie und quelloffene Software, die außerhalb einer Geschäftstätigkeit entwickelt oder bereitgestellt wird, fällt nicht unter die Cyberresilienz-Verordnung, und für *Verwalter quelloffener Software*, die solche Software für die kommerzielle Nutzung unterstützen, gilt eine leichtere Regelung. Reine Software as a Service fällt grundsätzlich nicht unter die Verordnung, es sei denn, sie ist die Lösung zur Datenfernverarbeitung eines Produkts; Cloud- und andere digitale Diensteanbieter können stattdessen unter die NIS-2-Richtlinie fallen. Ob ein bestimmtes Produkt in den Anwendungsbereich fällt und welches Konformitätsbewertungsverfahren gilt, ist eine Rechtsfrage für jede Organisation; die architektonische Folge ist, dass SBOMs, Schwachstellenbehandlung, Update-Mechanismen und Sicherheitsprotokollierung zu Nachweisen werden, die vorhanden sein müssen, statt guter Absichten.

HINWEIS

Information, keine Rechtsberatung

Die Beschreibungen der Executive Order, des OMB-Memorandums und der Cyberresilienz-Verordnung fassen zusammen, wie sich diese Maßnahmen auf die Softwarearchitektur auswirken. Sie sind keine Rechtsberatung; die Pflichten hängen vom Produkt, vom Markt und von der Rolle der Organisation ab und sollten anhand der amtlichen Texte und mit qualifizierten Beratern bestätigt werden.

Checkliste: Sicherheit der Lieferkette für ein Service-Team

Das meiste davon ist einmal eine Plattformfunktion und danach Gewohnheit.

- [ ] **Ein Inventar führen**
  Für jeden Build eine SBOM erzeugen und mit dem Artefakt speichern, damit „Sind wir betroffen?“ eine Abfrage ist und kein Projekt.
- [ ] **Abhängigkeiten festschreiben und verifizieren**
  Lockfiles committen, in der CI mit Hash-Prüfung installieren und den Build scheitern lassen, wenn Lockfile und Manifest nicht übereinstimmen.
- [ ] **Abhängigkeiten bewusst wählen**
  Vor dem Hinzufügen Wartung, Zahl der Maintainer, OpenSSF-Scorecard-Ergebnisse, Lizenz und den Umfang des transitiv mitgezogenen Codes prüfen.
- [ ] **Laufend aktualisieren, aber nicht sofort**
  Einen Bot jede Woche kleine Updates vorschlagen lassen; für Routine-Updates ein kurzes Mindestalter für Releases anwenden; Sicherheitskorrekturen beschleunigen.
- [ ] **Scannen, dann einordnen**
  Software Composition Analysis und Image-Scans in der CI und laufend in der Registry ausführen; nach Ausnutzbarkeit und Erreichbarkeit priorisieren, nicht nach Anzahl.
- [ ] **Build-Werkzeuge ebenso pinnen wie Bibliotheken**
  CI-Actions und Plug-ins auf vollständige Commit-Hashes und Basis-Images auf Digests pinnen; Änderungen daran wie Code prüfen.
- [ ] **Den Build isolieren und härten**
  Ephemere Runner, keine langlebigen Secrets (OIDC-Föderation nutzen), Tokens mit minimalen Rechten, geschützte Branches und Review durch zwei Personen für Release-Pfade.
- [ ] **Signieren und Provenienz erfassen**
  Artefakte mit Sigstore oder einem Gegenstück signieren, SLSA-Provenienz veröffentlichen und auf der Plattform Build L3 anstreben.
- [ ] **Beim Deployment verifizieren**
  Admission-Richtlinien weisen Images ab, die unsigniert sind, von der falschen Identität signiert wurden oder denen Provenienz aus dem erwarteten Repository fehlt.

- [ ] **Für das nächste Log4Shell planen**
  Üben, eine Bibliothek innerhalb eines Tages in jedem Service zu finden und zu patchen, und die eigenen Meldepflichten nach der Cyberresilienz-Verordnung, aus Verträgen und gegenüber Kunden kennen.

Häufige Fragen

Sollten wir alle Abhängigkeiten per Vendoring (Kopie) in unser Repository übernehmen?
Vendoring bietet Kontrolle und Offline-Builds, erschwert aber Updates und verdeckt Korrekturen aus dem Upstream. Ein Lockfile mit Hashes plus ein privater Registry-Mirror, der freigegebene Pakete zwischenspeichert, bietet den Großteil der Kontrolle mit weniger Wartung.

Reicht eine private Registry, um Dependency Confusion zu verhindern?
Nur wenn Builds so konfiguriert sind, dass sie interne Namen ausschließlich aus ihr auflösen. Verwenden Sie Paketnamen mit Scope oder Namespace, registrieren Sie Ihre internen Namen in öffentlichen Registrys und konfigurieren Sie den Paketmanager so, dass ein öffentliches Paket niemals ein internes überschreiben kann.

Wer in der Organisation ist für die Sicherheit der Lieferkette zuständig?
Meist ist sie geteilt: Ein Plattform- oder Sicherheitsteam stellt die gehärtete Pipeline, das Signieren, das Scannen und die Richtlinien einmal bereit; jedes Service-Team hält seine Abhängigkeiten wenige, aktuell und begründet.

PRÜFEN SIE IHR VERSTÄNDNIS

▼ **Warum haben die Signaturprüfungen die SolarWinds-Kunden nicht geschützt?**

Die Backdoor wurde während des eigenen Builds von SolarWinds eingefügt, sodass das trojanisierte Update tatsächlich von SolarWinds signiert war. Eine Signatur zeigt, wer ein Artefakt erzeugt hat, nicht, dass der Build sauber war.

▼ **Was ergänzt ein Lockfile gegenüber einem Manifest?**

Das Manifest gibt zulässige Versionsbereiche an; das Lockfile hält die exakt aufgelösten Versionen und meist ihre Hashes fest, sodass jeder Build dieselben Bytes installiert und ein verändertes Paket bei der Verifizierung durchfällt.

▼ **Warum ist der Verweis auf eine CI-Action per Tag riskanter als per Commit-Hash?**

Ein Tag kann von jedem verschoben werden, der das Repository kontrolliert, auch von einem Angreifer, sodass dieselbe Referenz unbemerkt anderen Code ausführen kann; ein vollständiger Commit-Hash bezeichnet unveränderlichen Inhalt.

▼ **Welche Frage beantwortet eine SBOM, die ein Schwachstellenscanner allein über eine ganze Systemlandschaft hinweg nicht schnell beantwortet?**

Welche deployten Artefakte in welchen Services und Umgebungen eine bestimmte Komponente in einer bestimmten Version enthalten, sodass sich die Betroffenheit von einer neu bekannt gegebenen Schwachstelle überall zugleich ermitteln lässt.

▼ **Probieren Sie es aus: Eine CI-Pipeline verwendet build-action@main und ein Basis-Image mit dem Tag latest und installiert Pakete ohne Lockfile. Niemand kann sagen, welche Services eine neu als verwundbar bekannte Bibliothek verwenden. Ordnen Sie die Korrekturen, die Sie vornehmen würden, nach Priorität, und begründen Sie jede.**

Erstellen Sie zuerst pro Build eine SBOM, damit sich eine neue Schwachstelle per Abfrage in Minuten finden lässt. Committen Sie als Nächstes Lockfiles mit Hashes, und pinnen Sie die CI-Action und das Basis-Image per Digest, denn Tags können sich verschieben, und wer sie ändern kann, erreicht Ihren Build. Automatisieren Sie dann kleine Abhängigkeits-Updates, mit kurzer Verzögerung bei neuen Releases. Signieren Sie schließlich Artefakte mit Provenienz, und prüfen Sie sie beim Deployment. Jeder Schritt begrenzt, was ein kompromittiertes Upstream anrichten kann.

QUELLEN

- SLSA specification v1.2: https://slsa.dev/spec/v1.2/
- Sigstore documentation: https://docs.sigstore.dev/
- Regulation (EU) 2024/2847 (Cyber Resilience Act), EUR-Lex: https://eur-lex.europa.eu/eli/reg/2024/2847/oj
- NVD: CVE-2024-3094 (xz utils backdoor): https://nvd.nist.gov/vuln/detail/CVE-2024-3094
- CycloneDX specification: https://cyclonedx.org/

Privacy by Design und die DSGVO

BEVOR SIE DAS LESEN

Dieser Abschnitt baut auf:

- 7.5 CQRS und Event-Sourcing
- 8.2 Bedrohungsmodellierung
- 8.4 Daten schützen

Begriffe von früher, die er verwendet: Crypto-Shredding, Datenresidenz, Datenflussdiagramm (DFD), Time to Live (TTL)

DIE IDEE IN KÜRZE

Stellen Sie sich vor, Sie leihen einem Freund Ihr Tagebuch. Sie erwarten, dass er nur die Seite liest, die Sie ihm gezeigt haben, sie nicht fotokopiert, es nicht für immer behält und es zurückgibt, wenn Sie darum bitten. Privacy by Design baut diese Erwartungen von Anfang an in Software ein: nur erheben, was nötig ist, es nur für den genannten Zweck verwenden, es nur so lange aufbewahren wie nötig und es ermöglichen, Menschen zu zeigen, was über sie gespeichert ist, und es auf Wunsch zu löschen. Gesetze wie die DSGVO machen aus diesen Erwartungen Pflichten, und viele davon lassen sich nur erfüllen, wenn das System dafür entworfen wurde.

WAS SIE WISSEN MÜSSEN

- Datenschutz fragt, ob Daten überhaupt erhoben und verwendet werden sollten und welche Kontrolle die Menschen behalten; ein System kann sicher sein und trotzdem den Datenschutz verletzen.
- Die DSGVO (Verordnung (EU) 2016/679) und die UK GDPR mit dem Data Protection Act 2018, geändert durch den Data (Use and Access) Act 2025, teilen die Grundsätze aus Artikel 5: Rechtmäßigkeit, Verarbeitung nach Treu und Glauben, Transparenz, Zweckbindung, Datenminimierung, Richtigkeit, Speicherbegrenzung, Integrität und Vertraulichkeit sowie Rechenschaftspflicht.
- Artikel 25 verlangt Datenschutz durch Technikgestaltung (geeignete Maßnahmen wie die Pseudonymisierung) und durch datenschutzfreundliche Voreinstellungen (nur die erforderlichen Daten werden verarbeitet, gespeichert und zugänglich gemacht) und stützt sich auf die sieben Prinzipien von Ann Cavoukian.

- Die Rechte auf Auskunft, Berichtigung, Löschung, Einschränkung der Verarbeitung, Datenübertragbarkeit und Widerspruch erfordern eine Datenlandkarte, Export- und Lösch-Handler je Service, Einschränkungsflags und Sperrlisten; Antworten erfolgen in der Regel innerhalb eines Monats.
- Eine DSFA ist vor einer Verarbeitung mit voraussichtlich hohem Risiko erforderlich (Artikel 35) und gelingt am besten zusammen mit dem Bedrohungsmodell auf Basis desselben Datenflussdiagramms.
- Pseudonymisierte Daten bleiben für jeden, der sie re-identifizieren kann, personenbezogene Daten; eine Anonymisierung muss allen Mitteln standhalten, die nach allgemeinem Ermessen wahrscheinlich genutzt werden, und ist schwer zu erreichen.
- Die Löschung muss Caches, Suchindizes, Analytics, Event-Logs (oft per Crypto-Shredding), Auftragsverarbeiter und Backups erreichen (bis zu ihrem Ablauf der Nutzung entzogen); dieser Abschnitt informiert darüber, wie sich das Recht auf die Architektur auswirkt, und ist keine Rechtsberatung.

HINWEIS

Information, keine Rechtsberatung

Dieser Abschnitt erklärt, wie sich das Datenschutzrecht auf Softwarearchitektur auswirkt. Er fasst die Rechtslage zum Zeitpunkt der Erstellung zutreffend zusammen, ist aber keine Rechtsberatung: Ob und wie eine Vorschrift gilt, hängt vom Sachverhalt ab, und Organisationen sollten sich von qualifizierten Fachleuten für Datenschutzrecht und ihrem Datenschutzbeauftragten beraten lassen.

Datenschutz ist eine Designeigenschaft

Sicherheit und Datenschutz überschneiden sich, sind aber nicht dasselbe. Sicherheit fragt, ob Daten vor Personen geschützt sind, die sie nicht haben sollten. Datenschutz fragt außerdem, ob die Organisation selbst die Daten überhaupt erheben sollte, was sie damit tun darf, wie lange und welche Kontrolle die Person behält, die sie beschreiben. Ein System kann frei von Sicherheitsverletzungen sein und trotzdem in die Privatsphäre eingreifen, etwa indem es den Standortverlauf jedes Kuriers für immer speichert oder Bestellhistorien stillschweigend zum Training eines Preismodells wiederverwendet.

Die Essenslieferplattform zeigt gut, warum Datenschutz eine Architekturfrage ist. Sie speichert Namen, Telefonnummern und Wohnadressen; Bestellhistorien, die über Ernährungsgewohnheiten Religion oder Gesundheit offenbaren können; Zahlungsreferenzen; und einen fortlaufenden Strom von Kurierstandorten. Diese Daten fließen von den Apps durch Dutzende von Services, Events, Caches, Logs, Analytics-Speichern und Backups und zu

Auftragsverarbeitern wie dem Zahlungsanbieter, dem SMS-Gateway und dem Support-Desk-SaaS. Entscheidungen darüber, wo personenbezogene Daten gespeichert werden, über welche Schlüssel sie adressiert werden und wie sie sich finden und entfernen lassen, sind später teuer zu ändern, und genau das ist der Test für architektonische Bedeutsamkeit.

Das Recht in Kürze: DSGVO und UK GDPR

Die Datenschutz-Grundverordnung, Verordnung (EU) 2016/679, gilt seit dem 25. Mai 2018. Sie schützt personenbezogene Daten, die Artikel 4 Nummer 1 definiert als alle Informationen, die sich auf eine identifizierte oder identifizierbare natürliche Person beziehen, direkt oder indirekt, etwa mittels Zuordnung zu einem Namen, zu einer Kennnummer, zu Standortdaten oder zu einer Online-Kennung. Der Verantwortliche entscheidet über die Zwecke und Mittel der Verarbeitung und trägt die Hauptverantwortung; ein Auftragsverarbeiter verarbeitet im Auftrag des Verantwortlichen auf Grundlage eines Vertrags (Artikel 28), wie es das SMS-Gateway der Plattform tut. Die DSGVO gilt für Organisationen mit Niederlassung in der EU und nach Artikel 3 auch für andere, die Personen in der EU Waren oder Dienstleistungen anbieten oder ihr Verhalten beobachten.

Im Vereinigten Königreich wurde die DSGVO nach dem Brexit als UK GDPR beibehalten, die neben dem Data Protection Act 2018 gilt. Der Data (Use and Access) Act 2025 hat beide geändert; die meisten seiner datenschutzrechtlichen Änderungen sind am 5. Februar 2026 in Kraft getreten, darunter Änderungen bei den Rechtsgrundlagen, bei automatisierten Entscheidungen und bei internationalen Übermittlungen, und ab dem 19. Juni 2026 folgt eine Pflicht der Verantwortlichen, Datenschutzbeschwerden zu bearbeiten. Die Europäische Kommission hat ihre Angemessenheitsbeschlüsse für das Vereinigte Königreich im Dezember 2025 erneuert, sodass personenbezogene Daten weiterhin ohne zusätzliche Garantien aus der EU in das Vereinigte Königreich fließen können. Für Architekten haben beide Regelwerke dieselben Grundsätze, Rechte und Gestaltungspflichten; die Unterschiede sind Detailfragen für Juristen.

Die Höchstbußgelder für die schwersten Verstöße betragen nach der DSGVO 20 Mio. EUR oder 4 % des weltweiten Jahresumsatzes (nach der UK GDPR 17,5 Mio. GBP oder 4 %), je nachdem, welcher Betrag höher ist; häufigere Folgen sind allerdings Anordnungen, die Verarbeitung einzustellen, Reputationsschäden und die Kosten einer Neuentwicklung. Im November 2025 hat die Europäische Kommission im Rahmen eines *Digital Omnibus*-Pakets Änderungen der DSGVO vorgeschlagen; zum Zeitpunkt der Erstellung durchlaufen diese noch das Gesetzgebungsverfahren und ändern bisher nichts.

Die Grundsätze aus Artikel 5 und was sie für Systeme bedeuten

Prinzip	Kurz gesagt	Architektonische Folge
Rechtmäßigkeit, Verarbeitung nach Treu und Glauben, Transparenz (Art. 5 Abs. 1 Buchst. a)	Nur mit einer Rechtsgrundlage verarbeiten, so, wie Menschen es vernünftigerweise erwarten, und ihnen sagen, was Sie tun.	Eine Rechtsgrundlage je Zweck festhalten; Datenschutzhinweise aus derselben Datenlandkarte erzeugen, die die Systeme nutzen; kein verdecktes Tracking.
Zweckbindung (Art. 5 Abs. 1 Buchst. b)	Für festgelegte, eindeutige Zwecke erheben und Daten nicht für damit unvereinbare Zwecke weiterverwenden.	Daten mit ihren Zwecken kennzeichnen; die Kennzeichnung prüfen, bevor ein neuer Consumer (etwa eine Machine-Learning-Pipeline) sie lesen darf.
Datenminimierung (Art. 5 Abs. 1 Buchst. c)	Dem Zweck angemessen und erheblich sowie auf das notwendige Maß beschränkt.	Kein Geburtsdatum erheben, wenn Sie nur ein Ü18-Flag brauchen; den Standortverlauf des Kuriers nur so lange speichern, wie er benötigt wird.
Richtigkeit (Art. 5 Abs. 1 Buchst. d)	Sachlich richtig und erforderlichenfalls auf dem neuesten Stand.	Eine maßgebliche Quelle je Fakt; Menschen ihre Daten berichtigen lassen und Berichtigungen an Kopien weitergeben.
Speicherbegrenzung (Art. 5 Abs. 1 Buchst. e)	In identifizierbarer Form nicht länger als nötig gespeichert.	Eine Aufbewahrungsfrist für jeden Datenbestand, automatisch durchgesetzt durch TTLs, das Löschen von Partitionen und Ablaufjobs, auch in Logs und Backups.
Integrität und Vertraulichkeit (Art. 5 Abs. 1 Buchst. f)	Angemessene Sicherheit, einschließlich Schutz vor unbefugter Verarbeitung, Verlust und Schädigung.	Alles aus den Abschnitten 8.1 bis 8.6: Zugriffskontrolle, Verschlüsselung, Schlüsselverwaltung, Zero Trust, Kontrollen der Softwareware-Lieferkette.
Rechenschaftspflicht (Art. 5 Abs. 2)	Der Verantwortliche ist für die Einhaltung verantwortlich und muss sie nachweisen können.	Verzeichnisse von Verarbeitungstätigkeiten, DSFAs, Einwilligungsnachweise, Löschregister und Audit-Logs sind Nachweise, die die Architektur liefern muss.

Rechtsgrundlagen in Kürze

Artikel 6 erlaubt eine Verarbeitung nur auf einer von sechs Rechtsgrundlagen: Einwilligung, Erfüllung eines Vertrags mit der Person, rechtliche Verpflichtung, lebenswichtige Interessen, Wahrnehmung einer Aufgabe im öffentlichen Interesse oder berechtigte Interessen, sofern nicht die Interessen und Rechte der Person überwiegen. Besondere Kategorien von Daten, etwa Gesundheitsdaten oder religiöse Überzeugungen, brauchen zusätzlich eine Bedingung nach Artikel 9. Die Rechtsgrundlage wird je Zweck gewählt, nicht je System, und sie verändert, was die Architektur leisten muss. Auf der Essenslieferplattform wird die Lieferadresse zur Erfüllung des Vertrags verarbeitet; Rechnungen werden aufbewahrt, um eine rechtliche

Verpflichtung aus dem Steuerrecht zu erfüllen; die Betrugsprüfung stützt sich typischerweise auf berechtigte Interessen; und Marketing-E-Mails brauchen in der Regel eine Einwilligung, für die zusätzlich die Vorschriften zur elektronischen Werbung gelten.

Die Einwilligung hat die strengsten Mechanismen. Nach Artikel 7 muss der Verantwortliche nachweisen können, dass die Einwilligung erteilt wurde; sie muss sich auf einen bestimmten Zweck beziehen, und ihr Widerruf muss so einfach sein wie ihre Erteilung. Deshalb kann eine Einwilligung keine einzelne boolesche Spalte sein: Das System braucht einen Append-only-Nachweis darüber, wozu die Person eingewilligt hat, wann, über welchen Kanal und auf welchen Wortlaut hin, und jeder Consumer muss einen Widerruf zeitnah beachten.

```
{
  "event": "ConsentGiven",
  "subjectRef": "cust_4411",
  "purpose": "marketing_email",
  "noticeVersion": "privacy-notice-2026-09",
  "wording": "Send me offers from restaurants near me",
  "channel": "ios-app 7.3.0",
  "method": "unticked checkbox, user ticked",
  "at": "2026-10-03T18:22:41Z"
}
{
  "event": "ConsentWithdrawn",
  "subjectRef": "cust_4411",
  "purpose": "marketing_email",
  "channel": "email unsubscribe link",
  "at": "2026-10-19T07:05:12Z"
}
```

Einwilligung als Append-only-Folge von Events: Der aktuelle Stand ergibt sich aus dem jeweils letzten Event je Zweck, und die Historie ist der Nachweis, den Artikel 7 verlangt.

Datenschutz durch Technikgestaltung und durch datenschutzfreundliche Voreinstellungen

Artikel 25 macht Privacy by Design zur Rechtspflicht. Artikel 25 Absatz 1, Datenschutz durch Technikgestaltung, verlangt vom Verantwortlichen, sowohl zum Zeitpunkt der Festlegung der Mittel für die Verarbeitung als auch zum Zeitpunkt der eigentlichen Verarbeitung geeignete technische und organisatorische Maßnahmen wie die Pseudonymisierung zu treffen, die die Grundsätze wirksam umsetzen, unter Berücksichtigung des Stands der Technik, der Implementierungskosten, der Art und der Zwecke der Verarbeitung sowie der Risiken. Artikel 25 Absatz 2, durch datenschutzfreundliche Voreinstellungen, verlangt, dass durch Voreinstellung nur die für den jeweiligen bestimmten Zweck erforderlichen Daten verarbeitet werden; das gilt für die Menge der erhobenen Daten, den Umfang ihrer Verarbeitung, ihre

Speicherfrist und ihre Zugänglichkeit; insbesondere dürfen Daten nicht ohne Eingreifen der Person einer unbestimmten Zahl von Personen zugänglich gemacht werden. Die Leitlinien 4/2019 des Europäischen Datenschutzausschusses zu Artikel 25 geben praktische Beispiele.

Das Recht hat die Idee von Ann Cavoukian übernommen, der Information and Privacy Commissioner von Ontario, die Privacy by Design in den 1990er-Jahren entwickelte und 2009 ihre sieben Grundprinzipien veröffentlichte; die internationale Konferenz der Datenschutzbeauftragten hat sie 2010 gebilligt. Sie sind nach wie vor eine nützliche Checkliste für Design-Reviews.

Cavoukians sieben Grundprinzipien von Privacy by Design

Prinzip	Was es von einem Design verlangt
1. Proaktiv statt reaktiv; vorbeugend statt nachbessernd	Datenschutzschäden bereits beim Design vorhersehen, statt sie nach einem Vorfall zu beheben.
2. Datenschutz als Voreinstellung	Die Person muss nichts tun, um geschützt zu sein; optionales Teilen bleibt aus, bis sie es einschaltet.
3. Datenschutz ins Design eingebettet	Datenschutz ist Teil der Architektur und der Geschäftsprozesse, kein nachträglicher Anbau.
4. Volle Funktionalität: Positivsumme statt Nullsumme	Designs suchen, die sowohl den Datenschutz als auch das Geschäftsziel erreichen, statt eines dem anderen zu opfern.
5. Durchgängige Sicherheit: Schutz über den gesamten Lebenszyklus	Daten von der Erhebung bis zur sicheren Vernichtung schützen.
6. Sichtbarkeit und Transparenz	Praktiken sind offen und für Nutzer und unabhängige Prüfer überprüfbar.
7. Achtung der Privatsphäre der Nutzer	Nutzerzentriert bleiben: starke Voreinstellungen, klare Hinweise und leicht bedienbare Wahlmöglichkeiten.

Rechte, die die Architektur unterstützen muss

Bei den Rechten der betroffenen Personen wird Datenschutzrecht am unmittelbarsten zu Systemanforderungen. Ein Verantwortlicher muss in der Regel innerhalb eines Monats nach Eingang eines Antrags reagieren; bei komplexen oder zahlreichen Anträgen kann die Frist um weitere zwei Monate verlängert werden (Artikel 12 Absatz 3), und er muss die Daten der Person finden können, wo immer sie liegen. In einem System aus zwanzig Services, fünf SaaS-Auftragsverarbeitern und einem Data Warehouse ist das ohne gepflegte Datenlandkarte

unmöglich: das Verzeichnis von Verarbeitungstätigkeiten, das Artikel 30 ohnehin verlangt, so nah am Code geführt, dass es zutrifft. Nach den britischen Änderungen von 2025 muss ein Verantwortlicher bei der Beantwortung eines Auskunftsersuchens nur angemessene und verhältnismäßige Suchen durchführen, was eine gute Datenlandkarte schnell macht.

Die Rechte der betroffenen Personen und was jedes von Systemen verlangt

Recht	Was die Person verlangen kann	Was Systeme brauchen
Auskunft (Art. 15)	Bestätigung, dass Daten verarbeitet werden, eine Kopie sowie Informationen wie Zwecke, Empfänger, Speicherdauer und Herkunft.	Eine Datenlandkarte aller Speicher mit personenbezogenen Daten; eine Export-API je Service, adressiert über eine stabile Kunden-ID; Identitätsprüfung; ein Weg, das Ergebnis innerhalb der Frist zusammenzustellen und bereitzustellen.
Berichtigung (Art. 16)	Berichtigung unrichtiger und Vervollständigung unvollständiger Daten.	Eine maßgebliche Quelle je Feld; Berichtigungen per Events an Kopien und Caches weitergegeben; Empfänger informiert (Art. 19).
Löschung (Art. 17)	Löschung, wenn unter anderem die Daten nicht mehr benötigt werden oder die Einwilligung widerrufen wird, vorbehaltlich Ausnahmen wie rechtlicher Verpflichtungen.	Ein Lösch-Orchestrator, Löschregeln je Speicher (löschen, anonymisieren, mit Begründung aufbewahren), Crypto-Shredding für unveränderliche Speicher und ein Register, das den Abschluss belegt.
Einschränkung (Art. 18)	Daten bleiben gespeichert, werden aber sonst nicht verwendet, etwa solange die Richtigkeit bestritten wird.	Ein Einschränkungsflag, das jeder Consumer prüft, sodass Daten gespeichert, aber von Verarbeitungen wie Marketing oder Modelltraining ausgenommen sind.
Datenübertragbarkeit (Art. 20)	Von der Person bereitgestellte Daten, die auf Grundlage einer Einwilligung oder eines Vertrags automatisiert verarbeitet werden, in einem strukturierten, gängigen und maschinenlesbaren Format.	Ein Export in JSON oder CSV mit dokumentierten Feldern; optional die direkte Übermittlung an einen anderen Verantwortlichen, soweit technisch machbar.
Widerspruch (Art. 21)	Beendigung einer Verarbeitung auf Grundlage berechtigter Interessen oder einer öffentlichen Aufgabe, sofern keine zwingenden Gründe vorliegen; bei Direktwerbung uneingeschränkt.	Schalter je Zweck; eine Sperrliste (mit dem nötigen Minimum, etwa einer gehashten E-Mail-Adresse), damit die Person nicht durch einen späteren Import erneut aufgenommen wird.
Automatisierte Entscheidungen (Art. 22)	Schutzmaßnahmen gegen ausschließlich auf automatisierter Verarbeitung beruhende Entscheidungen mit rechtlicher Wirkung oder ähnlich erheblicher Beeinträchtigung.	Zum Beispiel die automatisierte Deaktivierung eines Kuriers: ein Weg zur Überprüfung durch einen Menschen, eine Erklärung und eine Möglichkeit, die Entscheidung anzufechten.

Datenschutz-Folgenabschätzungen

Artikel 35 verlangt eine Datenschutz-Folgenabschätzung (DSFA) vor einer Verarbeitung, die voraussichtlich ein hohes Risiko für die Rechte und Freiheiten von Personen zur Folge hat, insbesondere bei systematischem und umfassendem Profiling mit erheblichen Auswirkungen, bei umfangreicher Verarbeitung besonderer Kategorien von Daten oder von Daten über strafrechtliche Verurteilungen und Straftaten sowie bei systematischer umfangreicher Überwachung öffentlich zugänglicher Bereiche. Die Aufsichtsbehörden veröffentlichen Listen weiterer Fälle; die fortlaufende Standortverfolgung von Kurieren und die algorithmische Steuerung ihrer Arbeit sind typische Kandidaten. Eine DSFA beschreibt die Verarbeitung und ihre Zwecke, bewertet Notwendigkeit und Verhältnismäßigkeit, bewertet die Risiken für die Personen und hält die Abhilfemaßnahmen fest; bleibt ein hohes Risiko bestehen, muss der Verantwortliche vor Beginn die Aufsichtsbehörde konsultieren (Artikel 36).

Architekten sollten eine DSFA als Datenschutz-Gegenstück zum Bedrohungsmodell aus Abschnitt 8.2 behandeln, zur selben Zeit und auf Basis desselben Datenflussdiagramms erstellt. Das an der KU Leuven entwickelte LINDDUN-Framework leistet für Datenschutzbedrohungen, was STRIDE für Sicherheitsbedrohungen leistet, mit Kategorien wie Linking, Identifying, Detecting, Data Disclosure und Unawareness. Die Schlussfolgerungen der DSFA werden zu Architekturanforderungen wie *Standortverläufe werden 30 Tage nach der Lieferung gelöscht*, die sich testen lassen.

Internationale Übermittlungen und Datenresidenz

Kapitel V der DSGVO beschränkt Übermittlungen personenbezogener Daten in Länder außerhalb des Europäischen Wirtschaftsraums. Eine Übermittlung ist zulässig, wenn die Kommission ein angemessenes Schutzniveau des Landes festgestellt hat (Artikel 45), wenn geeignete Garantien wie die Standarddatenschutzklauseln der Kommission bestehen (Artikel 46) oder im Rahmen eng begrenzter Ausnahmen (Artikel 49). Nachdem der Gerichtshof den EU-US-Datenschutzschild (Privacy Shield) im Urteil Schrems II vom Juli 2020 für ungültig erklärt hatte, erfordern Übermittlungen auf Basis von Vertragsklauseln eine Prüfung des Rechts im Zielland; das EU-US Data Privacy Framework, dessen Angemessenheit im Juli 2023 festgestellt wurde, deckt Übermittlungen an US-Unternehmen ab, die sich danach zertifizieren, und hat im September 2025 eine erste Klage vor dem Gericht der EU überstanden (*Latombe/Kommission*), wobei weitere Verfahren, einschließlich eines etwaigen Rechtsmittels zum Gerichtshof, die Lage noch ändern könnten.

Daraus folgen zwei architektonische Punkte. Erstens ist eine Übermittlung nicht nur eine Datenbank in einer anderen Region: Fernzugriff durch ein Support-Team im Ausland, die Unterauftragsverarbeiter eines SaaS-Auftragsverarbeiters und ein anderswo gehosteter Logging-Dienst zählen ebenfalls, daher muss die Datenlandkarte festhalten, wo Daten

gespeichert sind und von wo aus auf sie zugegriffen wird. Zweitens lassen sich Anforderungen an die Datenresidenz, ob gesetzlich, vertraglich oder selbst gewählt, am einfachsten mit regionalen Deployment-Zellen erfüllen: Jede Region hält die personenbezogenen Daten und Schlüssel ihrer Kunden, globale Services sehen nur pseudonyme Referenzen, und die Replikation zwischen Regionen ist explizit und geprüft. Datenresidenz nachträglich in eine einzige globale Datenbank einzubauen, gehört zu den teuersten Änderungen, vor denen eine Plattform stehen kann; deshalb gehört sie zu den früh erfassten Randbedingungen.

Compliance außerhalb von EU und Vereinigtem Königreich: SOC 2, ISO 27001, HIPAA und CCPA

Wenn die Plattform an große Restaurantketten verkauft oder in neue Länder expandiert, tauchen weitere Namen auf. Dieser Abschnitt dient der Information und ist keine Rechtsberatung: Ob ein Gesetz gilt, hängt von den Umständen ab, und die Regeln ändern sich; klären Sie die aktuelle Rechtslage daher mit qualifizierter Beratung. Die Namen fallen in zwei Gruppen: Prüfverfahren, die Kunden verlangen, und Gesetze, die Sie binden.

SOC 2 ist ein Attestierungsbericht, weder eine Zertifizierung noch ein Gesetz. Nach den Standards des American Institute of CPAs (AICPA) prüft ein unabhängiger Prüfer die Kontrollen einer Dienstleistungsorganisation anhand von Kriterien in fünf Kategorien: Sicherheit, Verfügbarkeit, Verarbeitungsintegrität, Vertraulichkeit und Datenschutz. Sicherheit ist in jedem Bericht Pflicht, die Organisation ergänzt die weiteren Kategorien, die zu ihren Zusagen passen, und es gibt keine feste Liste vorgeschriebener Kontrollen. Ein **Type-1**-Bericht betrachtet die Ausgestaltung der Kontrollen zu einem Stichtag; ein **Type-2**-Bericht prüft zusätzlich, ob sie über einen Zeitraum, üblicherweise sechs bis zwölf Monate, wirksam funktionierten, und ist der häufigere von beiden. **ISO/IEC 27001:2022** (2024 um den Klimaschutz ergänzt) ist eine internationale Norm mit Anforderungen an ein Informationssicherheits-Managementsystem: eine risikobasierte Art, Sicherheit zu betreiben, mit 93 Referenzmaßnahmen in Anhang A. Anders als SOC 2 führt sie zu einem Zertifikat, das nur so nützlich ist wie sein angegebener Geltungsbereich.

HIPAA ist ein US-Bundesgesetz, und seine Security Rule (45 CFR Parts 160 and 164) verlangt administrative, physische und technische Schutzmaßnahmen für elektronische geschützte Gesundheitsinformationen. Sie bindet Covered Entities (Krankenversicherungsträger, Clearingstellen und viele Gesundheitsdienstleister) und deren Business Associates, nicht jeden, der Gesundheitsdaten hält. Die Plattform läge normalerweise außerhalb, doch die Lieferung von Rezepten für Apotheken wäre ein Grund, sich beraten zu lassen, ob sie zum Business Associate geworden ist. Das HHS schlug Ende 2024 strengere Regeln vor, darunter verpflichtende Verschlüsselung und Multi-Faktor-Authentifizierung; prüfen Sie, ob sie endgültig verabschiedet wurden.

Der **California Consumer Privacy Act** (CCPA) in der durch den California Privacy Rights Act geänderten Fassung erfasst gewinnorientierte Unternehmen, die in Kalifornien geschäftlich tätig sind und einen Schwellenwert erreichen: im Wesentlichen ein jährlicher Bruttoumsatz über einem auf $25 Millionen festgesetzten Betrag, der in jedem ungeraden Jahr an die Inflation angepasst wird ($26.625.000 ab Januar 2025; prüfen Sie den aktuellen Betrag), oder der Kauf, Verkauf oder die Weitergabe der personenbezogenen Daten von 100.000 oder mehr kalifornischen Einwohnern oder Haushalten oder die Erzielung von mindestens der Hälfte des Umsatzes aus deren Verkauf oder Weitergabe. Einwohner können Auskunft über ihre Daten verlangen, sie löschen und berichtigen lassen, dem Verkauf oder der Weitergabe für kontextübergreifende Werbung widersprechen und die Nutzung sensibler Daten einschränken. Seit Januar 2026 geltende Durchführungsvorschriften ergänzen Risikobewertungen und, in Phasen, Cybersicherheitsaudits. Andere US-Bundesstaaten haben eigene Datenschutzgesetze.

Die architektonische Lehre lautet, dass sich die Anforderungen überschneiden: Least Privilege, Verschlüsselung, Logging, getestete Wiederherstellung und ein Dateninventar kehren in allen wieder. Bauen Sie einen Satz von Kontrollen und ordnen Sie ihn jedem Verfahren zu, statt für jedes ein eigenes Programm aufzusetzen, und verkleinern Sie den Geltungsbereich durch Datenklassifizierung und Segmentierung, wie Abschnitt 8.4 für PCI DSS gezeigt hat.

Kontinuierliche Compliance-Nachweise

Ein Type-2-Bericht fragt, ob eine Kontrolle das ganze Jahr funktioniert hat; Screenshots, die in der Woche vor dem Audit gesammelt werden, sind also schwach und teuer. Besser ist es, Nachweise als Nebenprodukt der normalen Arbeit zu sammeln: Pipeline-Logs, die zeigen, dass jedes Release Review und Scan bestanden hat, Infrastrukturdefinitionen, Zugriffsüberprüfungen, der Audit-Trail (Abschnitt 8.8). Jeder Nachweis wird mit den Kontrollen verknüpft, die er stützt, und eine Kontrolle, die versagt, löst im Moment des Versagens einen Alarm aus, nicht erst Monate später. Abschnitt 11.4 beschreibt die Seite von Policy as Code. Die Open Security Controls Assessment Language (OSCAL) des NIST bietet maschinenlesbare XML-, JSON- und YAML-Formate für Kontrollkataloge, Implementierungsdetails und Bewertungsergebnisse und ersetzt Dokumente durch strukturierte Daten. Automatisierung sammelt Nachweise; sie ersetzt weder das Urteil des Prüfers noch die Menschen, die für die Kontrollen verantwortlich sind.

Minimierung, Pseudonymisierung und Anonymisierung

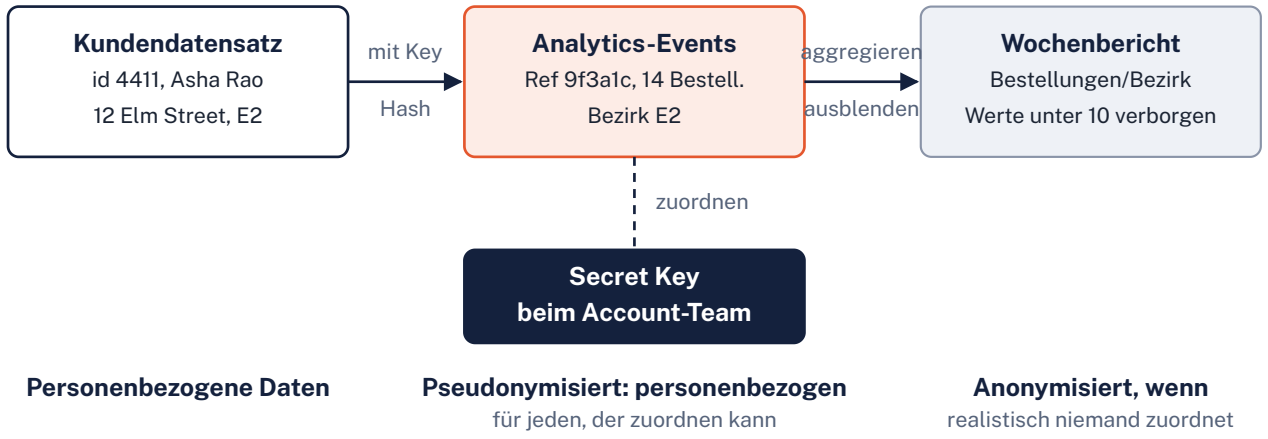
Die am billigsten zu schützenden personenbezogenen Daten sind die, die Sie nie erheben. Minimierung zeigt sich in kleinen Entscheidungen: ein Ü18-Flag statt eines Geburtsdatums, ein Postleitzahlbezirk statt einer vollständigen Adresse für Analytics, eine maskierte

Telefonnummer, die über die Plattform weitergeleitet wird, statt dem Kurier die Nummer des Kunden preiszugeben, und Standortabfragen nur während einer laufenden Lieferung.

Die Pseudonymisierung (Artikel 4 Nummer 5) verarbeitet Daten so, dass sie ohne Hinzuziehung zusätzlicher Informationen, etwa eines Schlüssels, die gesondert aufbewahrt und geschützt werden, nicht mehr einer Person zugeordnet werden können. Ein Keyed Hash (HMAC) der Kunden-ID, dessen Schlüssel beim Account-Team liegt, erlaubt dem Analytics-Team, Bestellungen je Kunde zu zählen, ohne zu sehen, wer die Kunden sind. Sie ist eine wertvolle Schutzmaßnahme, die Artikel 25 ausdrücklich nennt, doch Erwägungsgrund 26 stellt klar, dass pseudonymisierte Daten, die durch Heranziehung zusätzlicher Informationen einer Person zugeordnet werden könnten, weiterhin personenbezogene Daten sind. In *EDSB/SRB* (Rechtssache C-413/23 P, 4. September 2025) hat der Gerichtshof eine Nuance ergänzt: Ob Daten personenbezogen sind, wird aus der Sicht dessen beurteilt, der sie hält; pseudonymisierte Daten sind daher für einen Empfänger, der keine vernünftigen Mittel zur Re-Identifizierung hat, möglicherweise keine personenbezogenen Daten, bleiben es aber für den Verantwortlichen, der den Schlüssel hält, und der Verantwortliche muss die Menschen weiterhin über solche Empfänger informieren. Die Leitlinien 01/2025 des EDSA zur Pseudonymisierung beschreiben, wie man sie gut umsetzt.

Die Anonymisierung geht weiter: Die Daten beziehen sich nicht mehr auf eine identifizierbare Person, unter Berücksichtigung aller Mittel, die nach allgemeinem Ermessen wahrscheinlich genutzt werden, um jemanden zu identifizieren, und dann gilt die DSGVO nicht mehr. Echte Anonymisierung ist schwer. Namen zu entfernen genügt nicht, wenn die Kombination aus Postleitzahl, Geburtsdatum und Geschlecht die meisten Menschen identifiziert oder wenn der Standortverlauf eines einzelnen Kuriers verrät, wo er wohnt. Aggregation mit Unterdrückung kleiner Fallzahlen, Generalisierung und für Statistiken Differential Privacy sind die üblichen Techniken, und das Ergebnis sollte gegen realistische Re-Identifizierungsversuche getestet werden.

Personenbezogene, pseudonymisierte und anonymisierte Daten



Pseudonymisierung senkt das Risiko, doch die Daten bleiben für jeden personenbezogen, der sie zurückführen kann; eine Anonymisierung muss allen Mitteln zur Re-Identifizierung standhalten, die nach allgemeinem Ermessen wahrscheinlich genutzt werden.

WENN DAS NEU FÜR SIE IST

Schritt für Schritt: verborgene Namen sind nicht anonym

Der schwierigste Gedanke dieses Abschnitts ist der Unterschied zwischen pseudonymisierten und anonymisierten Daten. Beide verbergen Namen, doch wie der Abschnitt erklärt, behandelt die DSGVO sie sehr unterschiedlich. Verfolgen Sie eine Studie zur Sicherheit von Kurieren.

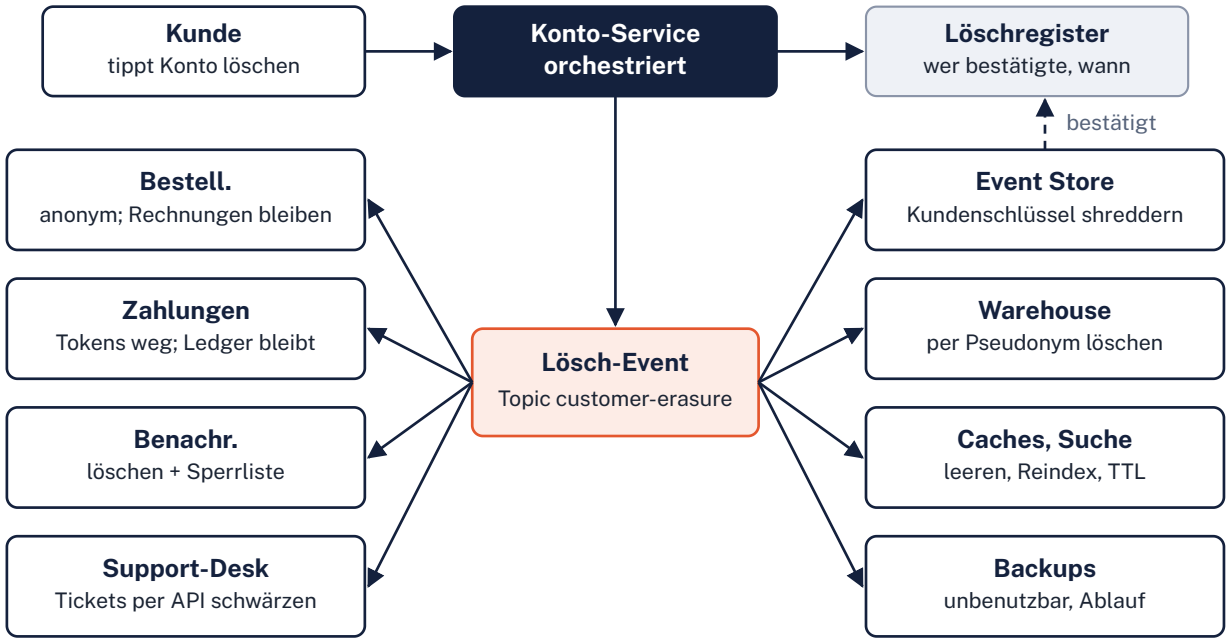
- Ein Kurierdatensatz enthält eine ID, einen Namen und eine Telefonnummer: eindeutig personenbezogene Daten.
- Für eine Studie zur Verkehrssicherheit ersetzt die Plattform die Kurier-ID durch einen Keyed Hash: einen Referenzcode, erzeugt mit einem geheimen Schlüssel, den das Kurier-Operations-Team gesondert aufbewahrt. Die Forschenden sehen den Code, eine Lieferanzahl und die Stadtzone, keine Namen oder Telefonnummern.
- Wer den Schlüssel hat, kann aus einer Kurier-ID denselben Code bilden und die Zeilen dieses Kuriers finden. Die Daten sind pseudonymisiert: Namen sind verborgen, aber Personen lassen sich weiterhin herausgreifen. Da die Plattform den Schlüssel hält, gilt die Regel des Abschnitts: Die Daten bleiben für die Plattform personenbezogen, ihre Forschenden eingeschlossen.
- Es lohnt sich trotzdem: Die Forschenden können Arbeitslasten vergleichen, ohne zu sehen, wer wer ist.
- Als Nächstes erstellt die Plattform einen Monatsbericht der Lieferungen je Stadtzone und lässt Zonen mit weniger als zehn Kurieren weg. Könnte niemand mit Mitteln, die nach allgemeinem Ermessen wahrscheinlich genutzt werden, jemanden daraus identifizieren, ist der Bericht anonymisiert, und wie der Abschnitt erklärt, gilt die DSGVO dann nicht mehr für ihn.
- Nur die Namen zu entfernen reicht oft nicht. Postleitzahl, Geburtsdatum und Geschlecht zusammen identifizieren die meisten Menschen, und der Standortverlauf eines Kuriers kann seine Wohnung verraten.
- **Die Lehre:** Ein verborgener Name ist keine anonyme Person. Pseudonymisierung senkt das Risiko, doch die Daten bleiben für jeden personenbezogen, der Personen re-identifizieren kann; nennen Sie Daten nur dann anonym, wenn niemand realistischerweise jemanden daraus re-identifizieren könnte, und testen Sie das mit realistischen Versuchen.

Aufbewahrung und Löschung über alle Kopien hinweg

Speicherbegrenzung braucht einen Aufbewahrungsplan: für jede Datenkategorie, wie lange sie aufbewahrt wird, warum und was am Ende geschieht (Löschung, Anonymisierung oder Archivierung). Pläne funktionieren nur, wenn Systeme sie automatisch durchsetzen: TTLs auf Cache- und Key-Value-Einträgen, zeitlich partitionierte Tabellen, deren alte Partitionen gelöscht werden, Lifecycle-Regeln im Objektspeicher, Aufbewahrungseinstellungen für Logs und geplante Jobs, deren Erfolg überwacht wird.

Am schwierigsten ist die Löschung dort, wohin personenbezogene Daten durchsickern. **Caches und Suchindizes** brauchen explizites Entfernen und Neuindizieren, mit TTLs als Absicherung. **Analytics**-Speicher sollten pseudonyme Referenzen halten, damit sich Zeilen finden und löschen lassen, oder die Daten sollten schon beim Eingang anonymisiert werden. **Event-Logs und Event-Sourcing-Speicher** sind von Natur aus unveränderlich; die übliche Technik ist Crypto-Shredding, bei dem die Felder jeder Person mit einem eigenen Schlüssel verschlüsselt werden und der Schlüssel vernichtet wird, wie in Abschnitt 7.5 beschrieben, oder personenbezogene Daten werden aus Events herausgehalten und nur Referenzen gespeichert. **Anwendungslogs** sollten keine personenbezogenen Daten enthalten, die nicht benötigt werden; darauf kommt Abschnitt 8.8 zurück. **Backups** lassen sich meist nicht bearbeiten; das britische Information Commissioner’s Office akzeptiert, dass Daten in Backups verbleiben, bis diese nach einem festgelegten Plan überschrieben werden, sofern sie bis dahin der Nutzung entzogen sind und die Person klar erfährt, was geschehen wird; wird ein Backup jemals wiederhergestellt, müssen die Löschungen erneut angewendet werden, wofür die Liste der gelöschten IDs aufbewahrt werden muss. Schließlich müssen **Auftragsverarbeiter** zur Löschung angewiesen werden, da sie nur auf Weisung des Verantwortlichen handeln (Artikel 28), und andere **Empfänger** müssen informiert werden (Artikel 19); die Löschung muss also über deren APIs das SMS-Gateway, den Support-Desk und die Marketingplattform erreichen.

Lösch-Fan-out auf der Essenslieferplattform



Jeder Speicher mit Daten des Kunden muss handeln und dies bestätigen

Ein Antrag, viele Speicher: Jeder Eigentümer wendet seine eigene Regel an (löschen, anonymisieren, shreddern oder mit Begründung aufbewahren) und bestätigt den Abschluss.

AUSGEARBEITETES BEISPIEL

Ausgearbeitetes Beispiel: „Konto löschen“ für die Essenslieferplattform entwerfen

Ein Kunde in London beantragt die Löschung seines Kontos. Das Design muss löschen, was gelöscht werden kann, mit angegebener Begründung behalten, was das Gesetz verlangt, deutlich innerhalb eines Monats fertig werden, Ausfälle mittendrin überstehen und Nachweise liefern. Die Aufbewahrungsfristen unten sind beispielhafte Festlegungen für dieses Beispiel, keine gesetzlichen Vorgaben für ein bestimmtes Unternehmen.

1. Prüfen und bestätigen	Der Antrag kommt in der App aus einer angemeldeten Sitzung; der Konto-Service verlangt eine erneute Authentifizierung, erklärt, was gelöscht und was behalten wird, und bietet zuerst einen Datenexport an. Eine kurze Bedenkzeit (zum Beispiel 14 Tage) erlaubt dem Kunden, den Antrag zurückzuziehen; offene Bestellungen und Erstattungen müssen zuerst abgeschlossen werden.
2. Den Antrag erfassen	Der Konto-Service schreibt einen Eintrag ins Löschregister mit einer Antrags-ID, der Kunden-ID und der Liste der Speicher, von denen eine Antwort erwartet wird, entnommen aus der Datenlandkarte. Das Register speichert keine weiteren personenbezogenen Daten als die ID, die zur Bearbeitung des Antrags nötig ist.
3. Fan-out	Er veröffentlicht `CustomerErasureRequested` über die Outbox an ein Topic. Jeder Consumer ist idempotent, da die Zustellung at least once erfolgt, und verarbeitet das Event auch dann, wenn der Kunde bereits teilweise gelöscht wurde.
4. Regeln je Speicher anwenden	Bestellungen ersetzen Name, Telefonnummer und Adresse durch einen Tombstone, behalten aber Bestellpositionen und Rechnungen für die steuerliche Aufbewahrungsfrist, mit der rechtlichen Verpflichtung als festgehaltener Begründung. Zahlungen bitten den Zahlungsanbieter, gespeicherte Karten-Tokens zu löschen, und behalten das Transaktions-Ledger. Benachrichtigungen löschen Kontaktdaten und fügen eine gehashte E-Mail-Adresse der Sperrliste hinzu. Der Event Store vernichtet den Datenschlüssel des Kunden. Das Warehouse löscht Zeilen per Pseudonym in seinem nächsten täglichen Lauf. Caches werden geleert und der Suchindex aktualisiert.
5. Auftragsverarbeiter erreichen	Konnektoren rufen die Lösch-APIs des Support-Desk-SaaS, des SMS-Gateways und der Marketingplattform auf und halten jede Antwort fest.
6. Bestätigen und nachfassen	Jeder Consumer schreibt eine Bestätigung ins Register. Ein täglicher Job löst einen Alarm für jeden Speicher aus, der nicht innerhalb von sieben Tagen bestätigt hat, deutlich innerhalb der Monatsfrist.

7. Backups behandeln	Datenbank-Backups werden 35 Tage aufbewahrt und ausschließlich für Disaster Recovery verwendet. Die gelöschten Kunden-IDs werden in einer Liste geführt, damit eine Wiederherstellung alle Löschungen erneut anwendet, bevor das System wieder geöffnet wird.
8. Den Kunden informieren	Eine abschließende Nachricht bestätigt den Abschluss, nennt, was behalten wurde, warum und bis wann, und dass Backup-Kopien innerhalb von 35 Tagen ablaufen.

Die Datenlandkarte ist das entscheidende Artefakt: Ein Speicher, der darin fehlt, ist ein Speicher, der die Daten behält. Eine Fitness Function kann den Build scheitern lassen, wenn ein neuer Service ein Feld mit personenbezogenen Daten deklariert, ohne einen Lösch-Handler zu registrieren.

Aufbewahrung mit Begründung ist kein Versagen der Löschung. Artikel 17 Absatz 3 nennt Ausnahmen, darunter die Erfüllung einer rechtlichen Verpflichtung; das Design behält das Minimum an Daten, das für diese Verpflichtung nötig ist, und löscht es, wenn die Verpflichtung endet.

Häufige Fragen

Reicht es, personenbezogene Daten zu verschlüsseln, um die DSGVO zu erfüllen?
Nein. Verschlüsselung ist eine wichtige Sicherheitsmaßnahme, und bei verschlüsselten Daten, die abfließen, müssen die Betroffenen unter Umständen nicht benachrichtigt werden, aber die Grundsätze, Rechtsgrundlagen, Rechte und Aufbewahrungsregeln gelten für verschlüsselte Daten wie für alle anderen.

Können wir alles in anonymisierter Form behalten, statt es zu löschen?
Nur wenn es wirklich anonym ist, also niemand realistischerweise Personen daraus re-identifizieren kann. Das Hashen einer ID oder das Weglassen von Namen ergibt meist pseudonymisierte Daten, die für jeden, der die Mittel zur Re-Identifizierung hat, weiterhin personenbezogen sind.

Wer legt die Aufbewahrungsfristen fest: Architekten oder Juristen?
Juristen und das Fachgeschäft legen die Fristen und Gründe fest; Architekten stellen sicher, dass jeder Speicher eine Frist hat, dass sie automatisch durchgesetzt wird und dass das System dies nachweisen kann.

PRÜFEN SIE IHR VERSTÄNDNIS

▼ Warum ist eine einzelne boolesche Spalte `marketing_consent` ein schwaches Design?

Sie kann nicht zeigen, wann, wie und zu welchem Wortlaut die Einwilligung erteilt wurde, was der Verantwortliche nach Artikel 7 nachweisen muss, und sie verliert die Historie der Widerrufe. Ein Append-only-Nachweis von Einwilligungs-Events liefert diesen Nachweis.

▼ Das Analytics-Team erhält Bestelldaten, die über einen HMAC der Kunden-ID adressiert sind. Sind das personenbezogene Daten?

Für die Plattform, die den Schlüssel hält, ja: Es sind pseudonymisierte Daten. Für einen Empfänger ohne vernünftige Mittel zur Re-Identifizierung sind sie es nach EDSB/SRB möglicherweise nicht, doch die Pflichten des Verantwortlichen bleiben bestehen.

▼ Warum gefährdet ein wiederhergestelltes Backup Löschungen, und was verhindert das?

Die Wiederherstellung bringt Daten zurück, die nach der Erstellung des Backups gelöscht wurden. Eine Liste der gelöschten IDs zu führen und die Löschungen erneut anzuwenden, bevor das wiederhergestellte System genutzt wird, verhindert das.

▼ Was meint Artikel 25 Absatz 2 mit Datenschutz durch datenschutzfreundliche Voreinstellungen?

Ohne jedes Zutun der Person werden nur die für den jeweiligen Zweck erforderlichen Daten erhoben, verarbeitet, gespeichert und zugänglich gemacht; zum Beispiel ist das optionale Teilen des Profils standardmäßig ausgeschaltet.

▼ Probieren Sie es aus: Das Marketing möchte die vollständige GPS-Spur jedes Kuriers „für Analysen“ für immer aufbewahren, mit den Namen der Kuriere. Wenden Sie Privacy by Design an: Was würden Sie infrage stellen, und welchen Entwurf würden Sie vorschlagen? Es geht hier um Architektur, nicht um Rechtsberatung.

Hinterfragen Sie zuerst den Zweck: Welche Frage braucht das? Speicherbegrenzung und Datenminimierung sprechen gegen „für immer“. Bewahren Sie genaue Verläufe nur so lange auf, wie Disposition und Streitfälle sie brauchen, und löschen Sie sie dann automatisch. Verwenden Sie für Analysen aggregierte Daten wie Geschwindigkeiten pro Zone, ohne Namen. Namen durch IDs zu ersetzen ist nur Pseudonymisierung, die Daten bleiben personenbezogen. Beschränken Sie den Zugriff standardmäßig, und führen Sie neben dem Threat Model eine DSFA durch, wenn die Standortverfolgung voraussichtlich ein hohes Risiko birgt.

QUELLEN

- Regulation (EU) 2016/679 (GDPR), EUR-Lex: https://eur-lex.europa.eu/eli/reg/2016/679/oj
- EDPB Guidelines 4/2019 on Article 25 Data Protection by Design and by Default: https://www.edpb.europa.eu/our-work-tools/our-documents/guidelines/guidelines-42019-article-25-data-protection-design-and_en
- ICO: Right to erasure: https://ico.org.uk/for-organisations/uk-gdpr-guidance-and-resources/individual-rights/individual-rights/right-to-erasure/
- SOC 2 services and report types (Schellman): https://www.schellman.com/soc-2
- An Introduction to SOC 2 (Accounting Today): https://www.accountingtoday.com/news/an-introduction-to-soc-2
- AICPA & CIMA: SOC suite of services: https://www.aicpa-cima.com/resources/landing/system-and-organization-controls-soc-suite-of-services
- ISO/IEC 27001:2022/Amd 1:2024: https://www.iso.org/standard/88435.html
- KPMG: ISO/IEC 27001:2022 overview: https://assets.kpmg.com/content/dam/kpmg/cn/pdf/en/2023/02/iso-iec-27001-2022.pdf
- HHS: Summary of the HIPAA Security Rule: https://www.hhs.gov/hipaa/for-professionals/security/laws-regulations/index.html
- HHS: HIPAA Security Rule NPRM fact sheet: https://www.hhs.gov/hipaa/for-professionals/security/hipaa-security-rule-nprm/factsheet/index.html
- California Attorney General: CCPA: https://oag.ca.gov/privacy/ccpa
- CPPA: Updated monetary thresholds in CCPA: https://www.cppa.ca.gov/regulations/cpi_adjustment.html
- CPPA: California finalizes regulations (23 September 2025): https://www.cppa.ca.gov/announcements/2025/20250923.html
- NIST OSCAL: https://pages.nist.gov/OSCAL/

Sicherheit im Betrieb: Logging, Erkennung und Reaktion

BEVOR SIE DAS LESEN

Dieser Abschnitt baut auf:

- 2.7 Sicherheit, Benutzbarkeit und mehr
- 8.2 Bedrohungsmodellierung
- 8.7 Privacy by Design

Begriffe von früher, die er verwendet: Personenbezogene Daten, Recovery Time Objective (RTO), Recovery Point Objective (RPO), Transactional Outbox, Software Bill of Materials (SBOM)

DIE IDEE IN KÜRZE

Eine Bank verlässt sich nicht nur auf starke Türen. Sie hat auch Kameras, die aufzeichnen, wer hereinkam, Alarmanlagen, die anschlagen, wenn etwas Ungewöhnliches geschieht, einen Plan, was das Personal bei einem Überfall tut, und Bargeldreserven, die anderswo eingeschlossen sind, falls der Haupttresor verloren geht. Software sicher zu betreiben funktioniert genauso. Systeme führen sorgfältige Aufzeichnungen wichtiger Aktionen, beobachten diese Aufzeichnungen auf Anzeichen eines Angriffs, beheben neu entdeckte Schwachstellen schnell, folgen einem eingeübten Plan, wenn etwas schiefgeht, halten Backups vor, an die Angreifer nicht herankommen, und informieren Aufsichtsbehörden und Kunden rechtzeitig, wenn personenbezogene Daten betroffen sind.

WAS SIE WISSEN MÜSSEN

- Gehen Sie davon aus, dass Sicherheitsverletzungen passieren: Entwerfen Sie für Erkennen, Eindämmen und Wiederherstellen, mit Sicherheitsszenarien, die messbare Reaktionszeiten festlegen.
- Protokollieren Sie Sicherheitsereignisse (Authentifizierung, Autorisierungsfehler, Admin- und Hochrisikoaktionen) mit wer, was, wann und wo; protokollieren Sie nie Passwörter, Tokens, Schlüssel, Kartendaten oder unnötige personenbezogene Daten; schützen Sie Logs mit getrenntem Write-once-Speicher.
- Ein Audit-Trail ist eine vollständige, unveränderliche, zurechenbare Aufzeichnung festgelegter Aktionen, ausgegeben vom zuständigen Service, wenn die Aktion committet wird.

- Ein SIEM korreliert Telemetrie und alarmiert nach Regeln, die aus dem Bedrohungsmodell und MITRE ATT&CK abgeleitet sind; Regeln sollten versioniert, getestet und so abgestimmt werden, dass keine Alarmmüdigkeit entsteht.
- Priorisieren Sie Schwachstellen nach bekannter Ausnutzung (KEV), Wahrscheinlichkeit (EPSS), Erreichbarkeit und Exposition, nicht allein nach dem CVSS-Basiswert, und vereinbaren Sie Behebungsziele im Voraus.
- NIST SP 800-61 Revision 3 (April 2025) ordnet Incident Response in die Funktionen des CSF 2.0 ein; Resilienz gegen Ransomware hängt von unveränderlichen Backups mit getrennten Zugangsdaten und getesteten Wiederherstellungen ab.
- Nach Artikel 33 DSGVO sind Verletzungen des Schutzes personenbezogener Daten unverzüglich und möglichst binnen 72 Stunden, nachdem sie bekannt wurden, der Aufsichtsbehörde zu melden und nach Artikel 34 den betroffenen Personen mitzuteilen, wenn das Risiko hoch ist.

Sicherheit endet nicht mit dem Deployment

Die früheren Abschnitte dieses Kapitels zielen darauf, Angriffe zu verhindern. Keine Prävention ist perfekt, daher entwerfen reife Organisationen auch unter der Annahme, dass eine Sicherheitsverletzung passieren wird: Sie wollen sie schnell bemerken, ihren Schaden begrenzen, sauber wiederherstellen und daraus lernen. Kapitel 2 beschrieb die Sicherheitstaktiken als Widerstehen, Erkennen, Reagieren und Wiederherstellen und zeigte, wie man Sicherheitsszenarien mit messbaren Reaktionen schreibt, etwa *ein abgeflossener Zugangsschlüssel wird innerhalb von 15 Minuten erkannt und widerrufen*. Dieser Abschnitt behandelt die Architektur und die Praktiken, die solche Reaktionen erreichbar machen.

Vieles davon baut auf den betrieblichen Grundlagen von Observability und Incident-Management in Kapitel 9 auf. Der Unterschied ist der Gegner: Eine ausfallende Festplatte verwischt keine Spuren, löscht nicht ihre Logs und kommt nicht nächste Woche durch eine Tür zurück, die sie offen gelassen hat.

Sicherheits-Logging: was protokolliert wird und was nie

Sicherheitslogs sollen im Nachhinein beantworten, wer was mit welcher Ressource getan hat, wann, von wo und mit welchem Ergebnis. Das OWASP Logging Cheat Sheet ist die Standardreferenz für die Praxis. Es empfiehlt, Ereignisse wie erfolgreiche und fehlgeschlagene Authentifizierungen, Autorisierungsfehler, Fehler im Sitzungsmanagement, Fehler bei der Eingabevalidierung, Anwendungsfehler, Hochrisikofunktionen wie administrative Aktionen und Datenexporte sowie die Nutzung der Sicherheitskontrollen selbst festzuhalten. Jedes Ereignis

sollte wann (ein synchronisierter Zeitstempel), wo (Anwendung, Service, Host, Umgebung), wer (die authentifizierte Identität, die Quelladresse) und was (die Aktion, das Objekt, das Ergebnis und der Grund) enthalten.

Ebenso wichtig ist, was draußen bleiben muss. Logs werden weit kopiert, lange aufbewahrt und von vielen Menschen und Tools gelesen, daher sind sie ein beliebter Ort, aus dem Secrets und personenbezogene Daten abfließen. Protokollieren Sie nie Passwörter, Sitzungskennungen oder Access Tokens (protokollieren Sie einen Hash oder eine Key-ID, wenn Sie korrelieren müssen), Verschlüsselungsschlüssel und Verbindungszeichenfolgen, Zahlungskartendaten oder personenbezogene Daten besonderer Kategorien. Behandeln Sie Logdaten als personenbezogene Daten, wo sie Personen identifizieren, und wenden Sie die Minimierungs- und Aufbewahrungsregeln aus Abschnitt 8.7 an. Schwärzung wird am besten zentral in der Log-Pipeline und zusätzlich im Code durchgesetzt, weil eine einzige vergessene Debug-Anweisung sie aushebeln kann.

Das protokollieren, das nie (nach dem OWASP Logging Cheat Sheet)

Festhalten	Ausschließen, maskieren oder hashen
Erfolgreiche und fehlgeschlagene Authentifizierungen, MFA-Abfragen, Passwort-Resets	Passwörter, auch falsche (ein vertipptes Passwort ist oft fast das richtige)
Autorisierungsfehler (Zugriff verweigert), besonders wiederholte	Sitzungskennungen, Access Tokens, Refresh Tokens, API-Schlüssel (stattdessen einen Hash oder die Key-ID protokollieren)
Fehler bei der Eingabevalidierung und vom Gateway abgelehnte Anfragen	Verschlüsselungsschlüssel, Datenbank-Verbindungszeichenfolgen und andere Secrets
Administrative Aktionen: Rollenänderungen, Richtlinienänderungen, Benutzeranlage, Konfigurationsänderungen	Vollständige Zahlungskartennummern, Kartenprüfnummern, Bankverbindungsdaten
Geschäftsaktionen mit hohem Risiko: Erstattungen, Auszahlungen, Datenexporte, Kontolöschung	Gesundheitsdaten, staatliche Kennnummern und andere personenbezogene Daten besonderer Kategorien
Sicherheitsrelevante Fehler und Exceptions; Start und Stopp von Sicherheitskontrollen	Personenbezogene Daten über das für Untersuchungen Nötige hinaus und alles, dem ein Nutzer widersprochen hat
Nutzung von Break-Glass- oder Notfallzugängen	Quellcode und Daten, die höher eingestuft sind, als das Logging-System speichern darf

```
{
  "ts": "2026-10-21T19:42:07.318Z",
  "event": "authz.denied",
  "service": "orders",
  "env": "prod",
  "actor": {"type": "user", "id": "cust_4411", "session": "sha256:7c1e..."},
  "client": {"ip": "203.0.113.24", "ua_family": "iOS app 7.3"},
  "action": "GET /orders/81723",
  "resource_owner": "cust_9902",
  "reason": "not_owner",
  "trace_id": "4bf92f3577b34da6a3ce929d0e0e4736"
}
```

Ein strukturiertes Sicherheitsereignis aus dem Bestellservice. Es zeigt, wer was versucht hat und warum es abgelehnt wurde, mit einer Trace-ID zur Korrelation, aber ohne Token und ohne Kundennamen.

Logs müssen auch vor Angreifern geschützt werden, deren erster Schritt nach dem Eindringen oft darin besteht, die Beweise zu löschen. Senden Sie Sicherheitslogs zeitnah an ein getrenntes Konto oder einen getrennten Dienst, den die eigenen Zugangsdaten der Anwendung nicht ändern können; speichern Sie sie mit Write-once-Aufbewahrung; beschränken und protokollieren Sie den Zugriff darauf; und halten Sie Uhren synchron, damit Ereignisse aus verschiedenen Systemen zusammenpassen. Behandeln Sie Log-Inhalte als nicht vertrauenswürdige Eingaben: Ein Angreifer, der einen Zeilenumbruch in ein protokolliertes Feld einschleusen kann, kann gefälschte Einträge erzeugen (Log-Injection); kodieren oder verwerfen Sie daher Steuerzeichen und denken Sie daran, dass Log4Shell durch das Protokollieren von angreiferkontrolliertem Text ausgelöst wurde.

Audit-Trails

Ein Audit-Trail ist eine engere, strengere Art von Log: eine vollständige, chronologische, manipulationssichtbare Aufzeichnung festgelegter Aktionen, die geführt wird, um Rechenschaft herzustellen. Debug-Logs werden gesampelt, rotiert und geändert, wann immer Entwickler es wollen; ein Audit-Trail muss für seinen Geltungsbereich vollständig sein (jede Erstattung, jede Rollenänderung, jeder Zugriff auf den Support-Datensatz eines Kunden), unveränderlich, einer authentifizierten Identität zurechenbar und für einen festgelegten Zeitraum aufbewahrt. Vorschriften und Standards verlangen oft einen; PCI DSS etwa verlangt, die Audit-Log-Historie mindestens zwölf Monate aufzubewahren, wobei die letzten drei Monate sofort für Analysen verfügbar sein müssen.

Architektonisch sollten Sie den Audit-Trail als Produkt der Domäne entwerfen, nicht als Nebenprodukt des Loggings. Geben Sie Audit-Events von dem Service aus, dem die Aktion gehört, und zwar in dem Moment, in dem die Aktion committet wird (die Transactional Outbox garantiert, dass entweder beides geschieht oder keines), in einen Append-only-Speicher mit

eingeschränktem Schreibzugriff. Manipulationssichtbarkeit lässt sich durch Hash-Verkettung erreichen, bei der jeder Datensatz einen Hash des vorherigen enthält, oder indem regelmäßig ein signierter Digest anderswo verankert wird. Event-Sourcing-Systeme bekommen vieles davon geschenkt, wie Abschnitt 7.5 gezeigt hat. Die Control-Plane-Audit-Logs der Cloud-Anbieter, die jeden API-Aufruf an das Cloud-Konto aufzeichnen, gehören zum selben Bild und müssen in jedem Konto und jeder Region aktiviert sein.

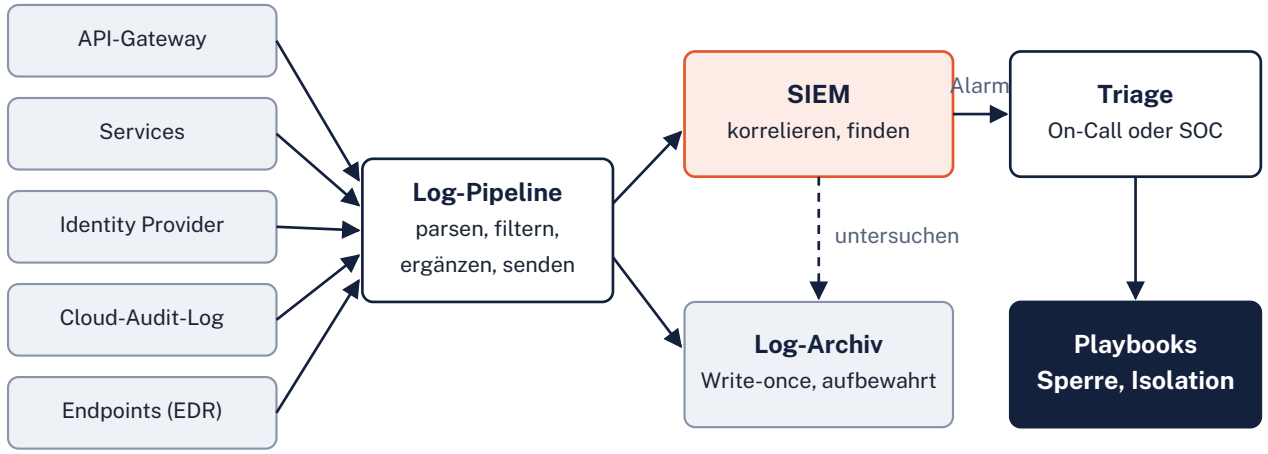
Erkennung: SIEM, Regeln und Reaktionsautomatisierung

Ein Security-Information-and-Event-Management-System (SIEM) sammelt Sicherheitstelemetrie aus der gesamten IT-Landschaft, normalisiert sie, korreliert Ereignisse und löst Alarme aus, wenn Muster zu Erkennungsregeln passen: viele fehlgeschlagene Anmeldungen, gefolgt von einer erfolgreichen, ein Servicekonto, das aus einem neuen Land genutzt wird, eine Rolle, die außerhalb eines Änderungsfensters Administratorrechte erhält, ein großer Export um drei Uhr morgens. Typische Quellen sind das API-Gateway, Sicherheitsereignisse der Anwendungen, der Identity Provider, Cloud-Audit-Logs, Kubernetes-Audit-Logs und Agents für Endpoint Detection and Response (EDR) auf Laptops und Servern der Mitarbeitenden.

Gute Erkennung wird entwickelt, nicht gekauft. Leiten Sie Regeln aus dem Bedrohungsmodell und aus MITRE ATT&CK ab, einer öffentlichen Wissensbasis über Taktiken und Techniken realer Angreifer, die MITRE seit 2015 pflegt; schreiben Sie sie als Code in einem gemeinsamen Format wie Sigma, versionieren Sie sie und testen Sie sie, indem Sie den Angriff simulieren, den sie erkennen sollen. Messen Sie die Alarmqualität: Eine Regel, die fünfzigmal am Tag auslöst und immer harmlos ist, bringt Menschen bei, sie zu ignorieren, und über Alarmmüdigkeit werden echte Angriffe übersehen. Günstige Stolperdrähte mit starkem Signal helfen: Honeytokens, etwa ein gefälschter Zugangsschlüssel oder ein gefälschter Kundendatensatz, den kein legitimer Prozess je anfasst, erzeugen fast keine Fehlalarme. Tools für Security Orchestration, Automation and Response (SOAR) führen dann Playbooks für gut verstandene Alarme aus, etwa einen Benutzer deaktivieren, einen Schlüssel widerrufen oder einen Container isolieren, sodass Menschen sich um die Ermessensentscheidungen kümmern.

Die Aufnahme ins SIEM wird oft nach Volumen abgerechnet, was Teams dazu verleitet, Logs wegzulassen. Ein gestuftes Design hilft: Wertvolle Sicherheitsereignisse gehen zur Echtzeiterkennung ins SIEM; umfangreiche Logs gehen in günstigeren Write-once-Speicher, wo sie während einer Untersuchung durchsucht werden können.

Eine Pipeline zur Sicherheitserkennung



Erkennungsregeln sind Code: versioniert, an Beispielangriffen getestet und so justiert, dass jeder Alarm zählt

Telemetrie wird einmal gesammelt und geschwärzt und dann sowohl für die Echtzeiterkennung als auch, aus einem Write-once-Archiv, für Untersuchungen genutzt.

Schwachstellenmanagement

Jeden Tag werden neue Schwachstellen in der Software veröffentlicht, die Sie betreiben. Eine CVE-Kennung (Common Vulnerabilities and Exposures) wie CVE-2021-44228 gibt jeder öffentlich bekannten Schwachstelle einen eindeutigen Namen; das 1999 von MITRE gestartete Programm vergibt Kennungen über Hunderte von CVE Numbering Authorities, darunter große Hersteller und Open-Source-Stiftungen. Die US National Vulnerability Database ergänzt Analysen wie Kennungen der betroffenen Produkte und Bewertungen, doch ein Rückstau bei dieser Anreicherung ab 2024 und ein kurzer Schreck im April 2025, als der Vertrag zur Finanzierung des CVE-Programms beinahe auslief, zeigten das Risiko, von einer einzigen Quelle abzuhängen; die meisten Organisationen kombinieren heute mehrere Quellen, darunter die Advisory-Datenbank von GitHub und die Open-Source-Datenbank OSV.

CVSS (das Common Vulnerability Scoring System, gepflegt von FIRST; Version 4.0 erschien im November 2023) bewertet den Schweregrad von 0 bis 10, mit den Stufen niedrig (0,1 bis 3,9), mittel (4,0 bis 6,9), hoch (7,0 bis 8,9) und kritisch (9,0 bis 10,0). Der veröffentlichte Basiswert beschreibt die Schwachstelle allgemein; die Threat- und Environmental-Metriken von Version 4.0 passen ihn an die Ausnutzungsaktivität und an Ihre eigene Umgebung an. Ein Basiswert ist kein Maß für Ihr Risiko. Zwei Signale verbessern die Priorisierung erheblich: das Exploit Prediction Scoring System (EPSS, ebenfalls von FIRST), das die Wahrscheinlichkeit schätzt, dass eine Schwachstelle in den nächsten 30 Tagen ausgenutzt wird, und der Katalog der Known Exploited Vulnerabilities (KEV) der US Cybersecurity and Infrastructure Security Agency, der Schwachstellen mit Belegen für eine reale Ausnutzung auflistet. Kombinieren Sie

sie mit Erreichbarkeit (wird die verwundbare Funktion genutzt?) und Exposition (ist der Service aus dem Internet erreichbar?), und die meisten der Hunderte von Befunden eines Scanners landen in der Routinewarteschlange, während eine Handvoll noch heute Handeln verlangt.

Vereinbaren Sie Behebungsziele im Voraus, damit nicht jeder Befund zur Verhandlung wird, und messen Sie die Zeit bis zur Behebung. Die Werte unten sind typische Beispiele, kein Standard; regulierte Umgebungen können eigene vorgeben, PCI DSS etwa verlangt, kritische Sicherheitspatches innerhalb eines Monats nach Erscheinen zu installieren.

Beispielhafte Behebungsziele

Befund	Ziel
Bekanntermaßen ausgenutzt (zum Beispiel im KEV-Katalog der CISA), aus dem Internet erreichbar	Innerhalb von 24 bis 72 Stunden entschärfen; patchen, sobald ein Fix existiert
Kritischer Schweregrad, erreichbar, keine bekannte Ausnutzung	Innerhalb von 7 bis 14 Tagen
Hoher Schweregrad	Innerhalb von 30 Tagen
Mittlerer Schweregrad	Innerhalb von 90 Tagen oder mit dem nächsten Routine-Update
Niedriger Schweregrad oder nicht erreichbar	Routine-Updates; bei Aufschub den Grund festhalten

ENTSCHEIDUNGSHILFE

Entscheidungshilfe: Triage einer neu veröffentlichten Schwachstelle

Gehen Sie dies durch, wenn ein Scanner, ein Advisory oder eine Meldung eine Schwachstelle in etwas nennt, das Sie verwenden könnten.

Q1 **Enthält irgendein deploytes Artefakt die verwundbare Komponente und Version (prüfen Sie den SBOM-Speicher)?**

- **Ja** Weiter zur nächsten Frage und jeden betroffenen Service samt Verantwortlichem festhalten.
- **Nein** „Nicht betroffen“ mit den Belegen festhalten; weiter auf neue betroffene Versionen achten.

Q2 **Wird sie bekanntermaßen in freier Wildbahn ausgenutzt (zum Beispiel im KEV-Katalog der CISA gelistet), oder ist funktionierender Exploit-Code öffentlich?**

- **Ja** Notfallpfad: innerhalb von Stunden bis Tagen entschärfen (patchen, das Feature deaktivieren oder am Gateway oder an der WAF blockieren), dann ordentlich patchen.
- **Nein** Weiter zur nächsten Frage.

Q3 **Ist der verwundbare Code in Ihrer Nutzung erreichbar, und ist der Service aus dem Internet erreichbar oder verarbeitet er sensible Daten?**

- **Ja** Innerhalb des SLA für seinen Schweregrad beheben, angepasst mit den Environmental-Metriken von CVSS und mit EPSS.
- **Nein** Weiter zur nächsten Frage.

Q4 **Gibt es eine kompensierende Kontrolle, die die Ausnutzung vollständig verhindert (Feature deaktiviert, kein Netzwerkpfad)?**

- **Ja** Sie dokumentieren (eine VEX-Erklärung eignet sich dafür) und im routinemäßigen Update-Zyklus patchen.
- **Nein** Im routinemäßigen Zyklus patchen und neu bewerten, wenn eine Ausnutzung gemeldet wird.

WENN DAS NEU FÜR SIE IST

Schritt für Schritt: der höchste Wert kommt nicht immer zuerst

Die Idee, die hier die meiste Sorgfalt verlangt, ist, Schwachstellen nach dem eigenen Risiko statt nach dem Wert zu ordnen. Abschnitt 8.2 hat CVSS eingeführt: Sein Basiswert beschreibt eine Schwachstelle allgemein, nicht Ihr Risiko durch sie. Verfolgen Sie zwei Befunde, die am selben Morgen eintreffen.

- Befund A ist eine kritische Schwachstelle mit Basiswert 9,1 in einer Bibliothek, die im Bestellservice enthalten ist. Befund B ist eine hohe mit Basiswert 7,5 in Software, die das API-Gateway ausführt. Nach Wert sortiert kommt A zuerst.
- Die erste Prüfung ist, ob die Komponente tatsächlich deployt ist. Der SBOM-Speicher zeigt, dass beide es sind, und das Team hält jeden betroffenen Service und sein zuständiges Team fest.
- Als Nächstes kommt die bekannte Ausnutzung. B steht im Katalog der Known Exploited Vulnerabilities, den die CISA führt, eine US-Bundesbehörde: Es gibt Belege, dass Angreifer sie nutzen. Nichts deutet darauf hin, dass jemand A ausnutzt.
- Dann Erreichbarkeit und Exposition. Das Gateway ist dem Internet zugewandt, Angreifer können B also direkt erreichen. A liegt in einem Teil der Bibliothek, den der Bestellservice nie aufruft.
- B nimmt daher den Notfallpfad: innerhalb von Stunden bis Tagen entschärfen, etwa indem das betroffene Feature deaktiviert oder der Angriff am Edge blockiert wird, dann ordentlich patchen. A wird mit den Belegen als nicht erreichbar festgehalten und im routinemäßigen Update-Zyklus gepatcht; neu bewertet wird, wenn eine Ausnutzung gemeldet wird.
- Bei den vielen Befunden, die weder gelistet noch eindeutig unerreichbar sind, hilft EPSS bei der Rangfolge: Es schätzt, wie wahrscheinlich jeder in den nächsten 30 Tagen ausgenutzt wird.
- **Die Lehre:** Ordnen Sie Schwachstellen nach bekannter Ausnutzung, Wahrscheinlichkeit, Erreichbarkeit und Exposition, nicht allein nach dem Basiswert, und vereinbaren Sie Behebungsziele im Voraus, damit nicht jeder Befund neu verhandelt wird.

Penetrationstests, Bug-Bounties und Offenlegung

Ein Penetrationstest ist ein autorisierter, zeitlich begrenzter Versuch erfahrener Tester, in ein System einzudringen, idealerweise mit Kenntnis seiner Architektur und mit einem schriftlich vereinbarten Umfang. Er findet Klassen von Problemen, die Scanner übersehen, etwa fehlerhafte Autorisierung auf Objektebene und Missbrauch der Geschäftslogik (denselben Gutschein über eine Race Condition zweimal einlösen). Beauftragen Sie einen vor einem

großen Launch, nach erheblichen Architekturänderungen und regelmäßig; eine Red-Team-Übung geht weiter und testet Erkennung und Reaktion, indem sie über Wochen einen echten Gegner simuliert.

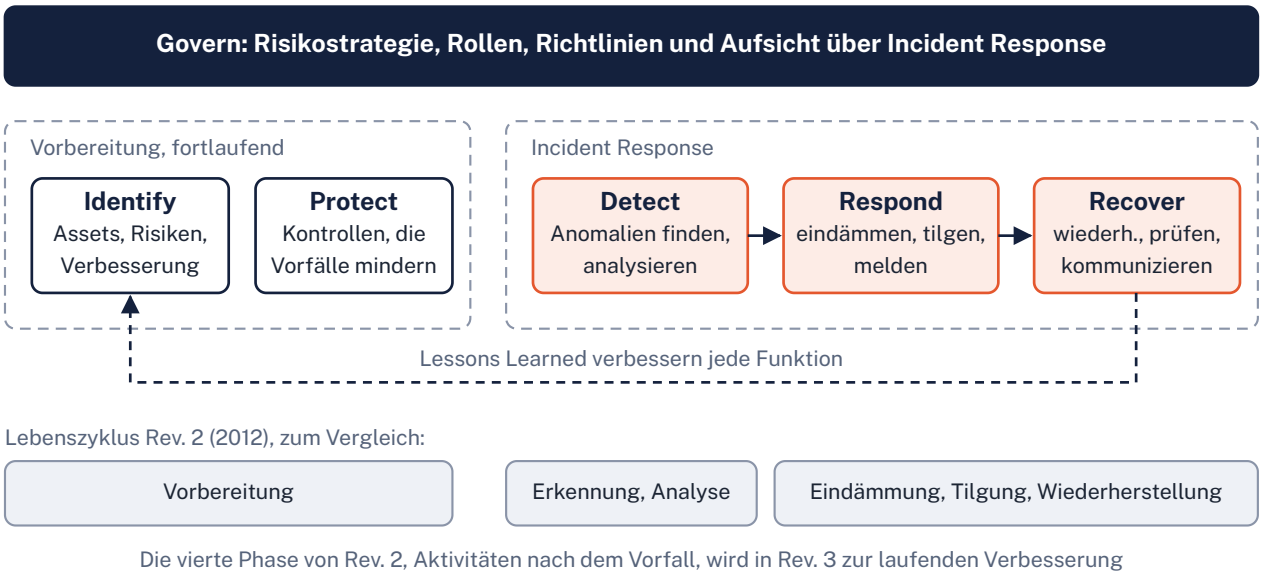
Externe Forschende werden ebenfalls Probleme finden und brauchen einen sicheren Weg, sie zu melden. Eine Vulnerability Disclosure Policy legt fest, wie gemeldet wird, welche Tests erlaubt sind und dass auf Forschung in gutem Glauben nicht mit rechtlichen Drohungen reagiert wird; eine Datei `security.txt` unter `/.well-known/security.txt` (RFC 9116, 2022) sagt Forschenden, wohin sie Meldungen schicken sollen. Ein Bug-Bounty ergänzt eine Bezahlung für gültige Befunde, oft über eine Plattform, die die Einreichungen sichtet. ISO/IEC 29147 (Offenlegung) und ISO/IEC 30111 (Bearbeitung) beschreiben die Prozesse, und die Cyberresilienz-Verordnung verpflichtet Hersteller von Produkten in ihrem Anwendungsbereich zu einer Strategie für die koordinierte Offenlegung von Schwachstellen. Ein Bounty ersetzt keine interne Sicherheitsarbeit: Auf einem ungetesteten System gestartet, erzeugt es meist eine Flut von Meldungen, die das Team nicht bewältigen kann.

Incident Response

Incident Response ist die organisierte Behandlung eines Sicherheitsvorfalls von der Erkennung bis zur Wiederherstellung und zum Lernen. Über ein Jahrzehnt lang war NIST SP 800-61 Revision 2 (2012) die Standardreferenz, mit ihrem Lebenszyklus aus vier Phasen: Vorbereitung; Erkennung und Analyse; Eindämmung, Beseitigung und Wiederherstellung; und Aktivitäten nach dem Vorfall. Revision 3, im April 2025 veröffentlicht als *Incident Response Recommendations and Considerations for Cybersecurity Risk Management: A CSF 2.0 Community Profile*, ersetzt diesen Lebenszyklus durch die sechs Funktionen des NIST Cybersecurity Framework 2.0. Govern, Identify und Protect sind fortlaufende Vorbereitung, die Vorfälle unwahrscheinlicher und die Reaktion wirksamer macht; Detect, Respond und Recover sind die Reaktion selbst; und Lessons Learned fließen in die Verbesserung aller Funktionen ein, statt eine letzte Phase zu sein.

In der Praxis braucht eine gute Reaktion benannte Rollen (einen Incident Commander, der koordiniert und entscheidet, technische Leads, die untersuchen und handeln, eine für Kommunikation verantwortliche Person, einen Protokollführer, der die Zeitleiste festhält, und schnellen Zugang zu Rechtsberatung und zum Datenschutzbeauftragten), Schweregrade, die festlegen, wer geweckt wird, und einen Out-of-Band-Kanal, weil ein Angreifer in Ihrem E-Mail- oder Chatsystem die Reaktion mitlesen kann. Eindämmung ist eine Abwägung: Einen Angreifer schnell abzuschneiden begrenzt den Schaden, kann aber Beweise vernichten und ihn warnen; sichern Sie daher Beweise (Snapshots, Logs, Speicherabbilder), bevor Sie neu aufbauen, und vergewissern Sie sich, dass Sie alle seine Brückenköpfe gefunden haben, bevor Sie verkünden, dass er weg ist. Danach fragt ein schuldfreies Review, wie das System und seine Prozesse den Vorfall ermöglicht haben, und erzeugt architektonische Maßnahmen.

Incident Response in NIST SP 800-61 Revision 3



Die vierte Phase von Rev. 2, Aktivitäten nach dem Vorfall, wird in Rev. 3 zur laufenden Verbesserung

Revision 3 bildet Incident Response auf die Funktionen des CSF 2.0 ab statt auf einen Lebenszyklus aus vier Phasen.

Resilienz gegen Ransomware

Ransomware verschlüsselt die Systeme einer Organisation und fordert Geld für den Schlüssel; die meisten Gruppen stehlen inzwischen vorher auch Daten und drohen, sie zu veröffentlichen. Die architektonischen Abwehrmaßnahmen sind die dieses ganzen Kapitels (MFA, Least Privilege, Segmentierung, Patchen, Erkennung), doch die Wiederherstellung hängt vor allem von einem ab: Backups, an die der Angreifer nicht herankommt. Angreifer mit Administratorrechten suchen zuerst nach Backups und löschen oder verschlüsseln sie.

Halten Sie mindestens eine Kopie kritischer Daten unveränderlich oder offline: Objektspeicher mit Write-once-Aufbewahrungssperren, die selbst ein Administrator nicht verkürzen kann, oder Medien, die physisch vom Netzwerk getrennt sind. Legen Sie Backups in ein getrenntes Konto oder einen getrennten Tenant mit eigenen Zugangsdaten und MFA, damit kompromittierte Produktionsidentitäten sie nicht erreichen. Die klassische 3-2-1-Regel (drei Kopien auf zwei Arten von Medien, eine davon außer Haus) ist eine vernünftige Grundlage. Testen Sie vor allem regelmäßig Wiederherstellungen gegen die RTO und RPO, die das Unternehmen vereinbart hat, einschließlich der Reihenfolge der Wiederherstellung: Identitätsdienste und DNS kommen zuerst, weil sich ohne sie nichts anderes wieder aufbauen lässt. 2017 baute die Reederei Maersk ihr weltweites Netzwerk neu auf, nachdem die Malware NotPetya ihre Windows-Domänencontroller zerstört hatte; die Wiederherstellung hing an dem einen Domänencontroller, der überlebt hatte, in Ghana, weil ein Stromausfall ihn während des Angriffs vom Netz genommen hatte.

Verletzungen des Schutzes personenbezogener Daten und Meldefristen

Eine Verletzung des Schutzes personenbezogener Daten ist nach Artikel 4 Nummer 12 DSGVO eine Verletzung der Sicherheit, die, ob unbeabsichtigt oder unrechtmäßig, zur Vernichtung, zum Verlust, zur Veränderung oder zur unbefugten Offenlegung von beziehungsweise zum unbefugten Zugang zu personenbezogenen Daten führt. Sie ist weiter gefasst als Diebstahl: Auch ein Ransomware-Angriff, der Daten unverfügbar macht, oder eine E-Mail an die falschen Empfänger kann darunter fallen. Nach Artikel 33 muss der Verantwortliche die Verletzung unverzüglich und möglichst binnen 72 Stunden, nachdem sie ihm bekannt wurde, der Aufsichtsbehörde melden, es sei denn, sie führt voraussichtlich nicht zu einem Risiko für die Rechte und Freiheiten von Personen; erfolgt die Meldung später, ist ihr eine Begründung für die Verzögerung beizufügen. Nach Artikel 34 muss der Verantwortliche außerdem die betroffenen Personen unverzüglich benachrichtigen, wenn die Verletzung voraussichtlich ein hohes Risiko für sie zur Folge hat. Artikel 33 Absatz 5 verlangt, jede Verletzung zu dokumentieren, ob gemeldet oder nicht. Die UK GDPR enthält dieselben Regeln, wobei Meldungen an das Information Commissioner’s Office gehen.

Andere Regelwerke können parallel gelten, mit eigenen Fristen, Empfängern und Schwellenwerten, und Verträge fügen oft Pflichten gegenüber Kunden und Partnern hinzu. Die Tabelle fasst die wichtigsten Fristen in der EU und im Vereinigten Königreich zusammen. Der Digital-Omnibus-Vorschlag der Europäischen Kommission vom November 2025 würde die Frist der DSGVO auf 96 Stunden verlängern und die Meldung an die Behörde auf Verletzungen mit hohem Risiko beschränken, doch zum Zeitpunkt der Erstellung ist er nicht angenommen, und es gilt die 72-Stunden-Regel.

Die architektonische Lehre ist, dass eine in Stunden gezählte Frist nur eingehalten werden kann, wenn die Systeme schnell und mit Belegen beantworten können, welche Daten betroffen waren, wem sie gehörten und ob tatsächlich darauf zugegriffen wurde. Das erfordert die Datenlandkarte aus Abschnitt 8.7, Datenzugriffslogs auf Speichern mit personenbezogenen Daten und eine Verschlüsselung, die eine abgeflossene Kopie unverständlich machen kann. Wie im Rest dieses Kapitels handelt es sich um Informationen darüber, wie sich das Recht auf Systeme auswirkt, nicht um Rechtsberatung; die Entscheidung über eine Meldung liegt beim Verantwortlichen, beraten durch seinen DSB und seine Juristen.

Meldefristen, die häufig gelten

Vorschrift	Wer meldet wem	Frist
DSGVO / UK GDPR Art. 33	Verantwortlicher an Aufsichtsbehörde (im Vereinigten Königreich das ICO)	Unverzüglich und möglichst binnen 72 Stunden, nachdem die Verletzung bekannt wurde, es sei denn, sie führt voraussichtlich nicht zu einem Risiko für Personen. Informationen können schrittweise bereitgestellt werden.
DSGVO / UK GDPR Art. 33 Abs. 2	Auftragsverarbeiter an Verantwortlichen	Unverzüglich, nachdem die Verletzung bekannt wurde.
DSGVO / UK GDPR Art. 34	Verantwortlicher an betroffene Personen	Unverzüglich, wenn die Verletzung voraussichtlich ein hohes Risiko zur Folge hat, es sei denn, die Daten waren zum Beispiel durch Verschlüsselung geschützt, die sie unverständlich macht.
NIS-2-Richtlinie (EU) 2022/2555	Wesentliche und wichtige Einrichtungen an CSIRT oder Behörde	Frühwarnung innerhalb von 24 Stunden, nachdem ein erheblicher Sicherheitsvorfall bekannt wurde, Meldung innerhalb von 72 Stunden, Abschlussbericht innerhalb eines Monats.
Cyberresilienz-Verordnung Art. 14	Hersteller, über die einheitliche Meldeplattform der ENISA	Für aktiv ausgenutzte Schwachstellen und schwerwiegende Sicherheitsvorfälle: Frühwarnung innerhalb von 24 Stunden, Meldung innerhalb von 72 Stunden (gilt ab dem 11. September 2026).

Runbooks

Ein Runbook ist ein schriftliches, getestetes Verfahren für eine bestimmte Situation: einen abgeflossenen Zugangsschlüssel widerrufen und rotieren, einen kompromittierten Container isolieren, die Bestelldatenbank aus einem unveränderlichen Backup wiederherstellen, auf eine vermutete Verletzung des Schutzes personenbezogener Daten reagieren. Ein gutes Runbook nennt seinen Auslöser, die einzubeziehenden Personen, die genauen Schritte und Befehle, wie sich prüfen lässt, dass jeder Schritt funktioniert hat, wie man zurückrollt und welche Beweise aufzubewahren sind. Runbooks sind dem Wesen nach Code: Halten Sie sie in der Versionskontrolle neben den Systemen, die sie beschreiben, automatisieren Sie die mechanischen Teile und üben Sie sie in Game Days und Tabletop-Übungen, weil ein Verfahren, das nie durchgespielt wurde, meist an Schritt drei scheitert. Jede Erkennungsregel sollte auf das Runbook für ihren Alarm verweisen.

Ausgearbeitetes Beispiel: die ersten 24 Stunden eines Vorfalls mit abgeflossenen Zugangsdaten auf der Essenslieferplattform

Zeit	Was geschieht	Warum sie zählt
09:12	Ein Engineer pusht einen Commit in ein öffentliches Repository, der einen langlebigen Cloud-Zugangsschlüssel für die Produktions-Deployment-Rolle enthält.	Die übliche Ursache: ein Debugging-Skript, keine böse Absicht. Automatisierte Scanner beobachten öffentliche Commits, daher muss sofort von einer Offenlegung ausgegangen werden.
09:14	Der Secret-Scanning-Alarm erreicht den Security-Bereitschaftsdienst; um 09:16 zeigt das SIEM API-Aufrufe mit dem Schlüssel von einer unbekannten Adresse.	Erkennung in Minuten setzt voraus, dass Secret Scanning und Cloud-Audit-Logs schon vor dem Vorfall ins SIEM fließen.
09:20	Vorfall mit höchstem Schweregrad ausgerufen. Ein Incident Commander, ein Protokollführer und ein technischer Lead werden benannt; ein eigener Kanal wird eröffnet.	Klare Rollen verhindern, dass fünf Leute dasselbe tun und niemand das Dringende.
09:25	Eindämmen: Der Schlüssel wird deaktiviert, mit ihm ausgestellte Sitzungen werden widerrufen, und eine Deny-all-Richtlinie wird an die Rolle gehängt, bis sie neu aufgebaut ist.	Deaktivieren statt Löschen erhält die Key-ID für Log-Suchen. Den Commit zu entfernen ist keine Eindämmung: Der Schlüssel wurde bereits kopiert.
09:45	Umfang: Das Team fragt das Cloud-Audit-Log nach jedem Aufruf mit der Key-ID ab, zurück bis zum Zeitpunkt ihrer Erstellung, nicht nur seit dem Push.	Der Schlüssel könnte schon früher auf anderem Weg abgeflossen sein; das Log beantwortet, was getan wurde, von wo und wann.
10:30	Befunde: Der Angreifer hat Storage-Buckets aufgelistet, 40 große Compute-Instanzen in einer ungenutzten Region gestartet und einen neuen Benutzer mit eigenem Zugangsschlüssel angelegt.	Angreifer sichern sich zuerst dauerhaften Zugang; nur den ursprünglichen Schlüssel zu bereinigen, würde sie im System lassen.
11:00	Beseitigen: Instanzen beendet, der neue Benutzer und sein Schlüssel gelöscht, kürzlich geänderte Richtlinien geprüft und zurückgesetzt; vorher Snapshots als Beweise gesichert.	Beweise sichern, bevor man sie vernichtet; sie werden für das Review, für Versicherer und möglicherweise für Aufsichtsbehörden gebraucht.
11:45	Datenfrage: Zugriffslogs auf Objektebene zeigen erfolgreiche Downloads aus dem Bucket mit den Kundenexporten. DSB und Rechtsberatung kommen hinzu.	Ohne Logs auf Objektebene könnte das Team nicht beweisen, dass keine Daten gelesen wurden, und müsste davon ausgehen, dass doch.

Zeit	Was geschieht	Warum sie zählt
12:30	Der DSB bewertet den Vorfall als Verletzung des Schutzes personenbezogener Daten, die voraussichtlich zu einem Risiko führt; die 72-Stunden-Frist wird ab dem Zeitpunkt der Kenntnis als laufend behandelt. Eine erste Meldung wird entworfen.	Artikel 33 erlaubt schrittweise Informationen; eine verspätete Meldung braucht eine Begründung. Ob die betroffenen Personen benachrichtigt werden müssen, hängt von der Höhe des Risikos ab.
15:00	Wiederherstellen: Deployment-Rolle mit kurzlebigen Zugangsdaten per OIDC-Föderation aus der CI neu aufgebaut; Abrechnungsalarme und Regionsbeschränkungen ergänzt; der Anbieter wegen betrügerischer Kosten kontaktiert.	Die Klasse des Problems beheben, nicht nur den Einzelfall: Ohne langlebige Schlüssel gibt es nichts, was abfließen kann.
18:00	Push-Schutz für Secrets in jedem Repository aktiviert; alle anderen Secrets, die vom Rechner desselben Engineers lesbar waren, vorsorglich rotiert.	Davon ausgehen, dass verwandte Zugangsdaten ebenfalls offengelegt sein könnten, bis das Gegenteil bewiesen ist.
Nächster Tag 09:00	Statusbericht an die Leitung; Meldung an die Aufsichtsbehörde innerhalb der 72 Stunden mit dem bekannten Stand eingereicht; schuldfreies Review innerhalb von fünf Arbeitstagen angesetzt.	Das Review fragt, warum ein langlebiger Schlüssel existierte und committet werden konnte, nicht, wer ihn committet hat.

Die Zeitleiste zeigt, warum der Betrieb Teil der Architektur ist. Jeder Schritt, der gut lief, hing von einer Designentscheidung ab, die Monate vorher getroffen wurde: Secret Scanning und Audit-Logs, die ins SIEM flossen, Zugriffslogs auf Objektebene für den Export-Bucket, eine Datenlandkarte, die dem DSB sagte, wessen Daten darin lagen, und ein Identitätsdesign, mit dem sich die Deployment-Rolle ohne langlebige Schlüssel neu aufbauen ließ. Jeder Schritt, der schlecht lief (dass es überhaupt einen langlebigen Schlüssel gab, ein Bucket mit Kundenexporten, den niemand mehr brauchte), war ebenfalls eine Architekturentscheidung.

Checkliste: Ist das System bereit, angegriffen zu werden?

- [ ] **Sicherheitsereignisse werden protokolliert, strukturiert und zentralisiert**
  Authentifizierung, Autorisierungsfehler, Admin- und Hochrisikoaktionen, mit wer, was, wann, wo und einer Trace-ID; keine Secrets und keine unnötigen personenbezogenen Daten.
- [ ] **Logs sind geschützt und werden aufbewahrt**
  Getrenntes Konto oder getrennter Speicher mit Write-once-Aufbewahrung, eingeschränktem Zugriff, synchronisierten Uhren und festgelegter Aufbewahrungsfrist.
- [ ] **Cloud-Audit- und Datenzugriffslogs sind aktiviert**
  Einschließlich Zugriffen auf Objektebene für Speicher mit personenbezogenen oder Zahlungsdaten, damit sich der Umfang beweisen lässt.
- [ ] **Für Ihre wichtigsten Bedrohungen gibt es Erkennungsregeln**
  Regeln, abgeleitet aus dem Bedrohungsmodell, mit simulierten Angriffen getestet, mit einem Verantwortlichen und einem Runbook für jeden Alarm.
- [ ] **SLAs für Schwachstellen sind vereinbart und werden gemessen**
  Priorisiert nach Ausnutzung, Erreichbarkeit und Exposition; der SBOM-Speicher beantwortet „Wo steckt es?“ in Minuten.
- [ ] **Es gibt einen Meldeweg für Schwachstellen**
  Eine Vulnerability Disclosure Policy und eine Datei security.txt; Penetrationstests vor großen Launches und nach großen Änderungen.
- [ ] **Der Incident-Response-Plan ist eingeübt**
  Rollen, Schweregrade, ein Out-of-Band-Kommunikationskanal und Kontakte zu Rechtsberatung, DSB, Versicherern und Anbietern; Übungen mindestens jährlich.
- [ ] **Backups überstehen einen Angreifer mit Admin-Rechten**
  Unveränderliche oder Offline-Kopien in einem getrennten Konto mit eigenen Zugangsdaten; Wiederherstellungen gegen RTO und RPO getestet.
- [ ] **Meldepflichten sind erfasst**
  Welche Aufsichtsbehörden, Kunden und Partner informiert werden müssen, von wem und innerhalb welcher Fristen, mit vorbereiteten Vorlagen.

PRÜFEN SIE IHR VERSTÄNDNIS

▼ Warum sollte ein Log einen Hash oder eine Key-ID statt des Access Tokens selbst festhalten?

Logs werden weit kopiert und gelesen, daher kann ein protokolliertes Token gestohlen und wiederverwendet werden. Ein Hash oder eine Key-ID erlaubt es Ermittelnden trotzdem, Ereignisse zu korrelieren, ohne nutzbare Zugangsdaten offenzulegen.

▼ Ein Scanner meldet eine Schwachstelle mit CVSS 9,8 in einer Bibliothek, deren verwundbare Funktion Sie nie aufrufen, und eine mit CVSS 7,5, die im KEV-Katalog gelistet ist, in Ihrem aus dem Internet erreichbaren Gateway. Welche zuerst?

Die im KEV gelistete Schwachstelle im Gateway: Sie wird bekanntermaßen ausgenutzt und ist exponiert. Die unerreichbare 9,8 kann dem routinemäßigen Patchen folgen, mit festgehaltener Begründung.

▼ Warum müssen Backups in einem getrennten Konto mit eigenen Zugangsdaten liegen?

Angreifer, die in der Produktion Administratorrechte erlangen, suchen nach Backups und zerstören sie; die Trennung bedeutet, dass kompromittierte Produktionsidentitäten die Wiederherstellungskopie nicht erreichen.

▼ Wann beginnt die 72-Stunden-Frist der DSGVO?

Wenn dem Verantwortlichen die Verletzung des Schutzes personenbezogener Daten bekannt wird, das heißt, wenn er mit hinreichender Sicherheit weiß, dass ein Sicherheitsvorfall personenbezogene Daten kompromittiert hat, nicht erst, wenn die Untersuchung abgeschlossen ist.

▼ Probieren Sie es aus: Schwachstelle A hat CVSS 9,8, liegt in einer Bibliothek auf einem internen Batch-Server, und ihre verwundbare Funktion wird nie aufgerufen. Schwachstelle B hat CVSS 7,5, betrifft Ihren öffentlichen Anmeldedienst und wird bekanntermaßen aktiv ausgenutzt. Welche kommt zuerst?

Zuerst B. Der CVSS-Basiswert allein ist ein schlechter Maßstab. B ist aus dem Internet erreichbar und steht auf der Liste bekannt ausgenutzter Schwachstellen, ein Angriff ist also wahrscheinlich und könnte einen kritischen Service treffen. Patchen Sie B jetzt, oder schirmen Sie die Schwachstelle ab, und prüfen Sie die Logs auf Anzeichen früherer Ausnutzung. A hat den höheren Wert, ist aber intern, und ihr Code ist nicht erreichbar, planen Sie sie also innerhalb der vereinbarten Frist ein. Dokumentieren Sie die Begründung, und prüfen Sie erneut, wenn sich die Gefährdung ändert.

QUELLEN

- OWASP Logging Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html
- NIST SP 800-61 Rev. 3: Incident Response Recommendations and Considerations (2025): https://csrc.nist.gov/pubs/sp/800/61/r3/final
- FIRST: Common Vulnerability Scoring System v4.0: https://www.first.org/cvss/v4-0/
- FIRST: Exploit Prediction Scoring System (EPSS): https://www.first.org/epss/
- RFC 9116: A File Format to Aid in Security Vulnerability Disclosure: https://www.rfc-editor.org/rfc/rfc9116

Weiterführende Literatur

- *Threat Modeling: Designing for Security*, Adam Shostack (2014), Wiley. Das Standardwerk für die Praxis der Bedrohungsmodellierung: die vier Fragen, Datenflussdiagramme, STRIDE, Angriffsbäume und wie man Bedrohungsmodellierung zum Teil der täglichen Entwicklungsarbeit macht.
- *Security Engineering: A Guide to Building Dependable Distributed Systems (3rd edition)*, Ross Anderson (2020), Wiley. Eine breite, gründlich recherchierte Darstellung, wie reale Systeme versagen, von Protokollen und Zugriffskontrolle bis zu Banken, Zahlungsverkehr und der Ökonomie der Sicherheit.
- *Designing Secure Software: A Guide for Developers*, Loren Kohnfelder (2021), No Starch Press. Security-by-Design-Prinzipien, Bedrohungsmodellierung und Reviews des sicheren Designs von einem Miterfinder von STRIDE, geschrieben für Entwickler und Architekten in der Praxis.
- *API Security in Action*, Neil Madden (2020), Manning. Authentifizierung, Tokens, OAuth 2.0, Capability-basierte Sicherheit und die Absicherung von Aufrufen zwischen Services, mit lauffähigem Code.
- *OAuth 2 in Action*, Justin Richer and Antonio Sanso (2017), Manning. Baut Clients, Autorisierungsserver und Ressourcenserver Schritt für Schritt auf, wodurch die Abläufe und die Angriffe darauf greifbar werden.
- *The Protection of Information in Computer Systems*, Jerome H. Saltzer and Michael D. Schroeder (1975), Proceedings of the IEEE, vol. 63, no. 9. Der Ursprung von Least Privilege, ausfallsicheren Standardwerten, vollständiger Zugriffsprüfung und den anderen Designprinzipien, die bis heute zitiert werden.
  https://doi.org/10.1109/PROC.1975.9939
- *SP 800-207: Zero Trust Architecture*, Scott Rose, Oliver Borchert, Stu Mitchell and Sean Connelly (NIST) (2020), National Institute of Standards and Technology. Die Referenzdefinition von Zero Trust, ihre Grundsätze und ihre logischen Komponenten (Policy Engine, Policy Administrator und Policy Enforcement Point).
  https://csrc.nist.gov/pubs/sp/800/207/final
- *Regulation (EU) 2016/679 (General Data Protection Regulation)*, European Parliament and Council of the European Union (2016), Official Journal of the European Union (EUR-Lex). Der Primärtext hinter dem Abschnitt über Datenschutz; für Architekten sind die Artikel 5, 25 und 32 bis 35 am wichtigsten.
  https://eur-lex.europa.eu/eli/reg/2016/679/oj

Glossar

Angriffsbaum
Ein Baum, dessen Wurzel das Ziel eines Angreifers ist und dessen Kinder die alternativen (ODER) oder kombinierten (UND) Wege dorthin sind, 1999 von Bruce Schneier bekannt gemacht.

Angriffsfläche
Die Gesamtheit der Stellen, an denen ein Angreifer versuchen kann, Daten in ein System einzubringen, Daten daraus abzuziehen oder anderweitig mit ihm zu interagieren: Endpoints, Ports, Dateien, Queues, Admin-Werkzeuge sowie die Menschen und der Code, denen es vertraut.

Anonymisierung
Die Verarbeitung von Daten in einer Weise, dass sie sich mit keinem Mittel, das vernünftigerweise eingesetzt würde, mehr auf eine identifizierbare Person beziehen; wirklich anonyme Daten fallen nicht unter die DSGVO.

Asymmetrische Kryptografie (Public-Key-Kryptografie)
Kryptografie mit einem Schlüsselpaar: einem öffentlichen Schlüssel, den jeder besitzen darf, und einem geheim gehaltenen privaten Schlüssel; verwendet für Schlüsselaustausch, digitale Signaturen und Zertifikate.

Attributbasierte Zugriffskontrolle (ABAC)
Zugriffsentscheidungen durch Auswertung einer Richtlinie über Attribute des Benutzers, der Ressource, der Aktion und des Kontexts, etwa Land, Datenklassifizierung oder Tageszeit.

Audit-Trail
Eine vollständige, chronologische und manipulationssichere Aufzeichnung festgelegter Aktionen, die zeigt, wer was wann getan hat, und der Nachvollziehbarkeit der Verantwortung dient.

Aufbewahrungsplan
Eine dokumentierte Liste, die festhält, wie lange jede Datenkategorie aufbewahrt wird, warum und was am Ende dieses Zeitraums mit ihr geschieht.

Auftragsverarbeiter
Eine Organisation, die personenbezogene Daten im Auftrag eines Verantwortlichen verarbeitet, nach dessen Weisungen und auf Grundlage eines Vertrags, etwa ein Cloud-Anbieter für SMS oder E-Mail.

Authentifizierung
Die Prüfung, ob ein Benutzer oder System der-oder dasjenige ist, das es zu sein behauptet, zum Beispiel durch ein Passwort, einen Passkey oder ein Zertifikat.

Autorisierung
Die Entscheidung, ob ein authentifizierter Benutzer oder ein authentifiziertes System eine bestimmte Aktion auf einer bestimmten Ressource ausführen darf.

Bedrohungsmodellierung
Eine strukturierte Art, zu einem Entwurf zu fragen, was wir bauen, was schiefgehen kann, was wir dagegen tun und ob wir gute Arbeit geleistet haben.

Beziehungsbasierte Zugriffskontrolle (ReBAC)
Zugriffsentscheidungen auf Grundlage von Beziehungen zwischen Benutzern und Objekten (Eigentümer von, Mitglied von, arbeitet bei), gespeichert als Graph; Googles Zanzibar (2019) ist das bekannteste Beispiel.

Bug-Bounty
Ein Programm, das externe Forschende für die Meldung gültiger Sicherheitslücken über ein vereinbartes Verfahren bezahlt.

CVE
Common Vulnerabilities and Exposures: das Programm, das jeder öffentlich bekannten Schwachstelle eine eindeutige Kennung wie CVE-2021-44228 gibt.

CVSS
Das von FIRST gepflegte Common Vulnerability Scoring System, das den Schweregrad einer bekannten Schwachstelle von 0 bis 10 bewertet; Version 4.0 wurde 2023 veröffentlicht.

Cyberresilienz-Verordnung (CRA)
Die Verordnung (EU) 2024/2847, die Cybersicherheitsanforderungen, den Umgang mit Schwachstellen und Meldepflichten für in der EU vertriebene Hardware- und Softwareprodukte festlegt.

Datenflussdiagramm (DFD)
Ein Diagramm aus externen Entitäten, Prozessen, Datenspeichern und den Datenflüssen zwischen ihnen, mit Vertrauensgrenzen quer über die Flüsse; das übliche Modell für die Bedrohungsmodellierung.

Datenschutz durch Technikgestaltung und durch datenschutzfreundliche Voreinstellungen
Die Pflicht nach Artikel 25 DSGVO, Datenschutzgrundsätze durch geeignete Maßnahmen in Systeme einzubauen und durch Voreinstellung nur die für den jeweiligen Zweck erforderlichen Daten zu verarbeiten.

Datenschutz-Folgenabschätzung (DSFA)
Eine nach Artikel 35 DSGVO erforderliche Abschätzung vor einer Verarbeitung, die voraussichtlich ein hohes Risiko für Personen zur Folge hat; sie beschreibt die Verarbeitung, ihre Notwendigkeit, die Risiken und die Abhilfemaßnahmen.

Defense in Depth
Das Schichten mehrerer unabhängiger Schutzmaßnahmen, sodass ein Angreifer, der eine überwindet, noch weiteren gegenübersteht, zum Beispiel eine Prüfung im Gateway, eine Prüfung im Service und die Verschlüsselung der Daten selbst.

Envelope Encryption
Daten mit einem Datenschlüssel verschlüsseln und diesen Datenschlüssel dann mit einem im KMS gehaltenen Key-Encryption-Key verschlüsseln (umhüllen), sodass Massendaten nie zum KMS übertragen werden und Schlüssel günstig rotiert werden können.

EPSS
Das Exploit Prediction Scoring System von FIRST, das die Wahrscheinlichkeit schätzt, dass eine Schwachstelle in den nächsten 30 Tagen ausgenutzt wird.

Hardware-Sicherheitsmodul (HSM)
Manipulationssichere Hardware, die Schlüssel in ihrem Inneren erzeugt und verwendet und sie nie im Klartext herausgibt; KMS-Dienste basieren meist auf HSMs.

Incident Response
Die organisierte Behandlung eines Sicherheitsvorfalls: ihn erkennen und analysieren, eindämmen, die Ursache beseitigen, den Betrieb wiederherstellen und daraus lernen.

JSON Web Token (JWT)
Ein kompaktes, signiertes (und optional verschlüsseltes) Token mit Claims wie Aussteller, Subjekt, Zielgruppe und Ablaufzeit, definiert in RFC 7519; weit verbreitet für Access Tokens und ID-Tokens.

Key Management Service (KMS)
Ein Managed Service, der kryptografische Schlüssel erzeugt, speichert, rotiert und ihre Verwendung kontrolliert und die Operationen selbst ausführt, sodass Anwendungen die Master-Schlüssel nie zu sehen bekommen.

Laterale Bewegung (Lateral Movement)
Das Vordringen eines Angreifers vom ersten kompromittierten Rechner oder Konto zu weiteren Systemen innerhalb desselben Netzwerks oder derselben Organisation.

Least Privilege
Das Prinzip, dass jeder Benutzer, jedes Programm und jeder Service nur die Berechtigungen erhält, die für die jeweilige Aufgabe nötig sind, und nur so lange wie nötig, sodass ein Fehler oder eine Kompromittierung nur begrenzten Schaden anrichtet.

Lockfile
Eine von einem Paketmanager erzeugte Datei, die die genauen Versionen und meist auch die Hashes aller aufgelösten Abhängigkeiten festhält, sodass jede Installation dieselben Pakete verwendet.

Mikrosegmentierung
Die Aufteilung eines Netzwerks in sehr kleine Zonen bis hinunter zu einzelnen Workloads, sodass jede nur Datenverkehr von den Aufrufern annimmt, die sie benötigt.

Multi-Faktor-Authentifizierung (MFA)
Authentifizierung, die zwei oder mehr verschiedene Arten von Faktoren verlangt: etwas, das Sie wissen, etwas, das Sie haben, und etwas, das Sie sind.

Mutual TLS (mTLS)
TLS, bei dem beide Seiten Zertifikate vorlegen und prüfen, sodass ein Service weiß, welcher Service ihn aufruft, und der Aufrufer zugleich den Server kennt.

OAuth 2.0
Das Framework für delegierte Autorisierung aus RFC 6749, bei dem ein Autorisierungsserver Access Tokens ausstellt, mit denen ein Client im Namen eines Ressourceneigentümers einen Ressourcenserver aufrufen kann, ohne dessen Passwort zu sehen.

OpenID Connect (OIDC)
Eine Identitätsschicht auf OAuth 2.0, die ein ID-Token hinzufügt, ein signiertes JWT, das dem Client mitteilt, wer der Benutzer ist und wie und wann er sich authentifiziert hat.

Passkey
Ein FIDO2/WebAuthn-Credential: ein für eine Website erzeugtes Schlüsselpaar, dessen privater Schlüssel auf dem Gerät oder im Passwortmanager des Benutzers bleibt, was die Anmeldung phishingresistent macht und kein gemeinsames Geheimnis auf dem Server hinterlässt.

Passwort-Hashing
Das Speichern eines Passworts als Ausgabe einer absichtlich langsamen, gesalzenen Funktion wie Argon2id, scrypt oder bcrypt, sodass eine gestohlene Datenbank nicht mit geringem Aufwand in Passwörter zurückverwandelt werden kann.

Perimetersicherheit
Das traditionelle Modell, das ein vertrauenswürdiges internes Netzwerk mit Firewalls und VPNs schützt und alles, was bereits innen ist, als freundlich behandelt; manchmal auch Burg-und-Graben-Modell genannt.

Personenbezogene Daten
Nach der DSGVO alle Informationen, die sich auf eine identifizierte oder identifizierbare natürliche Person (einen lebenden Menschen) beziehen, etwa ein Name, eine Kennnummer, Standortdaten oder eine Online-Kennung.

PKCE
Proof Key for Code Exchange (RFC 7636): Der Client sendet zu Beginn des Authorization-Code-Flows einen Hash eines Einmalgeheimnisses und beim Einlösen des Codes das Geheimnis selbst, sodass ein gestohlener Code nutzlos ist.

Policy Decision Point (PDP)
Die Komponente, die Zugriffsrichtlinien auswertet und mit „erlauben“ oder „verweigern“ antwortet; ein Policy Enforcement Point (PEP) fragt sie an und setzt die Antwort dann durch.

Policy Enforcement Point (PEP)
Die Komponente im Datenpfad, etwa ein Proxy, Gateway oder Sidecar, die jede Anfrage entsprechend einer Richtlinienentscheidung zulässt oder blockiert.

Provenienz (Provenance)
Überprüfbare Informationen darüber, woher ein Artefakt stammt: welche Build-Plattform es gebaut hat, aus welcher Quellcode-Revision, mit welchen Eingaben und Parametern.

Pseudonymisierung
Die Verarbeitung personenbezogener Daten in einer Weise, dass sie ohne Hinzuziehung gesondert aufbewahrter zusätzlicher Informationen nicht mehr einer Person zugeordnet werden können; nach der DSGVO sind pseudonymisierte Daten weiterhin personenbezogene Daten.

Public-Key-Infrastruktur (PKI)
Die Zertifizierungsstellen, Zertifikate, Trust Stores, Richtlinien und Sperrmechanismen, die es Beteiligten gemeinsam ermöglichen, die öffentlichen Schlüssel der jeweils anderen zu überprüfen.

Rechtsgrundlage
Eine der sechs Bedingungen in Artikel 6 DSGVO (Einwilligung, Vertrag, rechtliche Verpflichtung, lebenswichtige Interessen, öffentliche Aufgabe, berechtigte Interessen), auf der die Verarbeitung personenbezogener Daten beruhen muss.

Reproduzierbarer Build
Ein Build, der aus demselben Quellcode und denselben Eingaben bitgenau identische Ergebnisse erzeugt, sodass jeder neu bauen und prüfen kann, dass ein veröffentlichtes Artefakt nicht manipuliert wurde.

Rollenbasierte Zugriffskontrolle (RBAC)
Die Vergabe von Berechtigungen an Rollen und von Rollen an Benutzer, sodass der Zugriff über die Zuordnung von Personen zu Rollen wie Support-Mitarbeiter oder Restaurantleitung verwaltet wird.

Runbook
Eine schriftliche, erprobte Anleitung, mit der eine bestimmte Betriebs- oder Sicherheitssituation Schritt für Schritt bewältigt wird, einschließlich der Prüfung und des Rollbacks jedes Schritts.

Secrets-Management
Das Speichern, Verteilen, Rotieren und Auditieren von Zugangsdaten wie Passwörtern, API-Schlüsseln und privaten Schlüsseln in einem dedizierten Tresor (Vault) statt in Code oder Konfigurationsdateien.

Secure by Default
Die Auslieferung eines Produkts, dessen Standardkonfiguration die sichere ist, sodass Benutzer sich bewusst dafür entscheiden müssen, den Schutz abzuschwächen, statt daran denken zu müssen, ihn einzuschalten.

Secure Software Development Framework (SSDF)
NIST SP 800-218: eine Sammlung von Praktiken für sichere Entwicklung, gruppiert in: die Organisation vorbereiten, die Software schützen, gut gesicherte Software erstellen und auf Schwachstellen reagieren.

Shift Left
Die Verlagerung von Sicherheitsaktivitäten (oder Tests) auf frühere Phasen des Entwicklungslebenszyklus, in Anforderungen, Entwurf und Programmierung, wo Probleme billiger zu beheben sind als in der Produktion.

SIEM
Security Information and Event Management: ein System, das Sicherheitslogs aus vielen Quellen sammelt, korreliert und Alarme auslöst, wenn sie Erkennungsregeln entsprechen.

Sigstore
Ein Open-Source-Projekt zum Signieren und Verifizieren von Software-Artefakten mit kurzlebigen, an eine Identität gebundenen Zertifikaten und einem öffentlichen Transparenzlog, das langlebige Signierschlüssel überflüssig macht.

SLSA
Supply-chain Levels for Software Artifacts: ein Framework mit aufsteigenden Stufen der Gewissheit, dass ein Artefakt durch einen vertrauenswürdigen, manipulationssicheren Build aus dem erwarteten Quellcode erstellt wurde.

Software Bill of Materials (SBOM)
Ein maschinenlesbares Verzeichnis der Komponenten und Abhängigkeiten einer Software in einem Standardformat wie SPDX oder CycloneDX.

Software Composition Analysis (SCA)
Werkzeuge und Praktiken, die die Open-Source-Komponenten in Software ermitteln und mit Datenbanken bekannter Schwachstellen und Lizenzen abgleichen.

Software-Supply-Chain-Angriff
Ein Angriff, der etwas kompromittiert, das eine Organisation bezieht, etwa eine Abhängigkeit, ein Build-Werkzeug, ein Update oder eine Registry, um diese Organisation und weitere nachgelagerte zu erreichen.

STRIDE
Ein Merkwort für sechs Arten von Bedrohungen (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service und Elevation of Privilege), 1999 bei Microsoft von Loren Kohnfelder und Praerit Garg entwickelt.

Symmetrische Verschlüsselung
Verschlüsselung, bei der derselbe geheime Schlüssel ver- und entschlüsselt, etwa AES; schnell und daher für große Datenmengen verwendet.

Tokenisierung
Das Ersetzen eines sensiblen Werts wie einer Kartennummer durch ein bedeutungsloses Token, während der echte Wert in einem separaten, streng geschützten Tresor aufbewahrt wird, der oft von einem Zahlungsdienstleister betrieben wird.

Transitive Abhängigkeit
Ein Paket, das Ihre Software indirekt nutzt, weil eine Ihrer Abhängigkeiten oder eine von deren Abhängigkeiten darauf angewiesen ist.

Unveränderliches Backup
Eine Sicherungskopie, die vor Ablauf ihrer Aufbewahrungsfrist nicht geändert oder gelöscht werden kann, auch nicht von einem Administrator, und so vor Ransomware und böswilligen Insidern geschützt ist.

Verantwortlicher
Die Organisation, die über die Zwecke und Mittel der Verarbeitung personenbezogener Daten entscheidet und die Hauptverantwortung für die Einhaltung des Datenschutzrechts trägt.

Verletzung des Schutzes personenbezogener Daten
Nach Artikel 4 Nr. 12 DSGVO eine Verletzung der Sicherheit, die zur Vernichtung, zum Verlust oder zur Veränderung, ob unbeabsichtigt oder unrechtmäßig, oder zur unbefugten Offenlegung von beziehungsweise zum unbefugten Zugang zu personenbezogenen Daten führt.

Vertrauensgrenze
Eine Linie in einem System, an der Daten oder Kontrolle zwischen Teilen mit unterschiedlichem Vertrauens- oder Berechtigungsniveau wechseln, etwa vom Internet in eine API, sodass Eingaben dort geprüft werden müssen.

Workload Identity
Eine überprüfbare Identität für einen laufenden Service statt für eine Person, ausgestellt von der Plattform (zum Beispiel eine SPIFFE-ID in einem kurzlebigen Zertifikat), die statt gemeinsamer Geheimnisse verwendet wird.

Zero Trust
Ein Sicherheitsmodell, das kein implizites Vertrauen aufgrund des Netzwerkstandorts gewährt: Jede Anfrage wird pro Ressource und pro Sitzung authentifiziert, autorisiert und verschlüsselt (NIST SP 800-207).

Zertifizierungsstelle (CA)
Eine Organisation oder ein System, das digitale Zertifikate durch Signieren ausstellt und damit bestätigt, dass ein öffentlicher Schlüssel zu einem benannten Server, einer Person oder einem Workload gehört.