



DIE 15 SICHERHEITSPRINZIPIEN DER SOFTWAREENTWICKLUNG (TEIL 1)

📅 Freitag, 16.9.2022

🏷 IT-Sicherheit



Bei der Planung und anschließenden Umsetzung eines Software-Systems besteht der Anspruch, ein sicheres System zu entwickeln. Was jedoch zu tun ist, um diesem Anspruch gerecht zu werden, ist in der Praxis leider zu oft unklar. Daher widmen wir uns in einer **dreiteiligen Blogreihe der sicheren Softwareentwicklung** und stellen die 15 Prinzipien vor, die jede:r IT-Architekt:in und jede:r Software Entwickler:in kennen sollte. Als kleines Gimmick gibt es diese 15 Prinzipien auch als praktisches Cheat-Sheet. In diesem Blogpost definieren wir die Grundlage und erklären die ersten 6 Prinzipien.

echs



Haben Sie Fragen? Wir helfen gerne!

GRUNDLAGE: DIE SCHUTZZIELE DER IT-SICHERHEIT

Die primären Schutzziele der IT-Sicherheit sind **Vertraulichkeit**,



KONTAKT



BLOG



TERMIN





Nichtabstreitbarkeit, Zurechenbarkeit und Privatsphäre, die unterschiedlich stark je nach System zu berücksichtigen sind.

Aus den Schutzz Zielen haben wir Sicherheitsprinzipien abgeleitet, die bei dem Entwurf und der Implementierung von Systemen als Richtschnur für das Denken und Handeln dienen können. Warum diese 15 Sicherheitsprinzipien für den Entwurf und die Implementierung sicherer Software relevant sind und was jedes Prinzip uns sagen will, werden wir in diesem und folgenden Blogbeiträgen prägnant darstellen.

Inhaltsübersicht

1. KISS - Keep it simple, stupid
2. Positives Sicherheitsmodell
3. Ursachenbehebungsprinzip
4. Least Privilege
5. Vermeidungsprinzip
6. Minimalprinzip
7. Secure by Design ([in Teil 2 der Blogreihe](#))
8. Misstrauensprinzip ([in Teil 2](#))
9. Defense in Depth ([in Teil 2](#))
10. Fail Secure ([in Teil 2](#))
11. Kenne deinen Gegner ([in Teil 3 der Blogreihe](#))
12. Kerckhoffs'sches Prinzip ([in Teil 3](#))
13. Indirektionsprinzip ([in Teil 3](#))
14. Konsistente Sicherheit ([in Teil 3](#))
15. Nutzerfreundliche Sicherheit ([in Teil 3](#))

KISS (KEEP IT SIMPLE, STUPID)



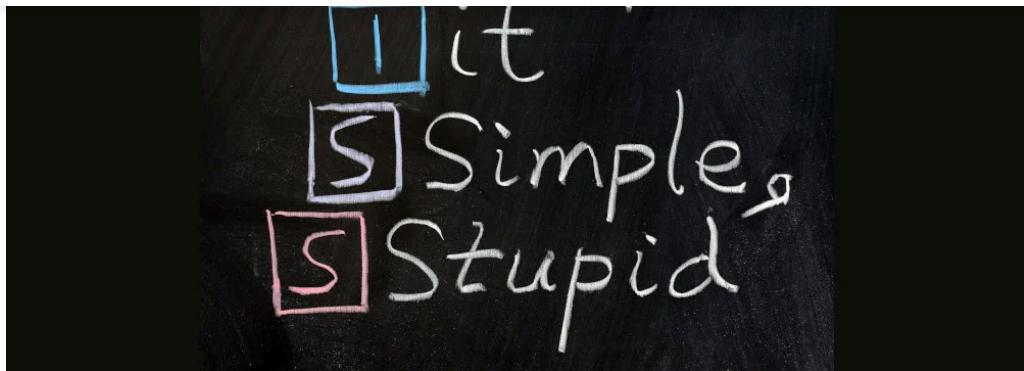
KONTAKT



BLOG



TERMINE



Dieses Prinzip, das unter dem Akronym KISS (Keep It Simple, Stupid) auch in jedem **Clean Code** Guide zu finden ist, gilt in besonderer Weise im Kontext der Sicherheit. Komplexität kann als Widersacher von Sicherheit gesehen werden. Deshalb sollte der Grundgedanke bei dem Design und der Implementierung von Sicherheitsfunktionen sein, sie möglichst einfach aber robust zu konzipieren und umzusetzen. Mit der steigenden Komplexität von Sicherheitsfunktionen, wächst ebenso die Gefahr, diese fehlerhaft zu verwenden, sei es in Bezug auf den Kontext oder die Konfiguration z.B. über Parameter.

WARUM SOLLTE ICH DIESES PRINZIP KENNEN / BERÜCKSICHTIGEN?

Als *Architekt:innen* eines Software-Systems können wir über klar verständliche und einfach konzipierte Sicherheitsfunktionen deren sichere und robuste Implementierung und Verwendung ermöglichen. Zudem können wir über die Auswahl intuitiv verwendbarer und leicht verständlichen Bibliotheken und Frameworks die Gefahr von Fehlverwendungen minimieren.

Als *Entwickler:innen* können wir Funktionalitäten des verwendeten Frameworks einsetzen. Das befreit uns davon komplexe Implementierungsdetails selbst umzusetzen und aufwändig pflegen zu müssen.

BEISPIELE FÜR DAS KISS-PRINZIP

Eine adäquate Verschlüsselung von Daten ist nicht zu unterschätzen. Je nach eingesetzter Bibliothek sind unterschiedlich viele Parameter anzugeben, die darüber bestimmen, welcher



Daten nicht sicher verschlüsselt werden (Beispiel: Electronic Code Book Mode). Ein positives Beispiel für eine gut dokumentierte und leicht zu verwendende Crypto-API ist [Google Tink](#).

Ein anderes Beispiel ist die gut gemeinte Absicherung von E-Mail-Adressen über reguläre Ausdrücke. Hier einen passenden regulären Ausdruck zu definieren, der einerseits den fachlichen Anforderungen entspricht und andererseits nicht für einen [Regular Expression Denial-of-Service](#)-Angriff genutzt werden kann, ist keine leichte Aufgabe. Aus Sicht des KISS-Prinzips kann hier eine sinnvolle Entscheidung zur Steigerung der Sicherheit sein, die Länge der E-Mail-Adresse zu begrenzen und nur druckbare Zeichen zu erlauben. Diese Prüfungen lassen sich in Java einfach mittels der `javax.validation-api` umsetzen.

VERWENDE EIN POSITIVES SICHERHEITSMODELL



An nahezu jedes Software-System sind aus fachlicher Sicht Anforderungen gerichtet, den Zugriff auf die Daten und deren Modifikation zu reglementieren. Grundsätzlich gibt es für die Umsetzung dieser Anforderung zwei Möglichkeiten: Blacklisting (Verbiete alles was nicht zulässig ist) oder Whitelisting (Erlaube alles was gewünscht ist). **Aus der Perspektive der Sicherheit sollte die**



KONTAKT



BLOG



TERMIN

WAKUM SOLLTE ICH DIESES PRINZIP KENNEN / BERÜCKSICHTIGEN?

Bei der Umsetzung von Berechtigungsprüfungen entscheiden sich Entwickler:innen und Architekt:innen zwangsläufig für eine der beiden genannten Vorgehensweisen, implizit oder besser explizit.

Bei der Wahl des positiven Sicherheitsmodells fällt es Entwickler:innen leichter, die Anforderungen erfüllen zu können.

Grundsätzlich hat so zunächst niemand Berechtigungen im System. Das führt dazu, dass die Funktion nicht verwendet werden kann.

Durch explizite Rechtevergabe werden Fachfunktionen dann freigeschaltet. Dies verhindert, dass versehentlich vergessen wird, die Verwendung einer Funktion zu verbieten. Wurde andersherum vergessen eine Berechtigung zu erteilen, so wird dies hingegen sehr wahrscheinlich im Test auffallen.

BEISPIELE FÜR DAS PRINZIP "POSITIVES SICHERHEITSMODELL"

Double Escaping ist eine Möglichkeit, um Sicherheitsmodelle die dem Blacklisting-Ansatz folgen auszuhebeln. Ein Beispiel ist der Zugriff auf beliebige Dateien des Webservers über die Anpassung der URL, d.h.

in <http://www.example.org/../../etc/passwd> wird die Zeichenfolge/ zunächst hexadezimal dargestellt (%2E%2E%2F), anschließend wird das %-Zeichen noch zusätzlich enkodiert (%252E%252E%252F). Der Security-Check prüft den Pfad auf/ und sollte den Request verwerfen. Vor dem Check wird jedoch nur die erste Enkodierung aufgelöst, so dass %2E%2E%2F nicht als fehlerhaft erkannt wird und der Request verarbeitet wird. Bei einem positiven Sicherheitsmodell wäre %2E%2E%2F nicht in der Liste der zulässigen Pfade enthalten, so dass der Request verworfen wird.

Das aktuelle Beispiel der Log4J-Lücke zeigt, dass man mit Blacklisting in einer Web Application Firewall nicht alle Enkodierungsmöglichkeiten abdecken kann (siehe Artikel bei [heise Schutz vor schwerwiegender Log4j-Lücke - was jetzt hilft und was nicht](#)).

Das Beispiel der Log4J-Lücke, deren Angriff über Blacklisting in einer Web Application Firewall abgefangen werden sollte führt uns auch



KONTAKT



BLOG



TERMIN



("URSACHENBEHEBUNGSPRINZIP")



Bei dem Bekanntwerden von Sicherheitslücken in einem Software-System ist immer die tatsächliche Ursache, also der Kern des Problems, zu beheben. Zwischenzeitliche Maßnahmen zur Mitigation eines Angriffsvektors können helfen, stellen allerdings nur ein Provisorium für die Zeit der Analyse und Behebung des Problems dar. Langfristig lässt sich über Workarounds aber kein sicherer Betrieb gewährleisten und die Sicherheitsarchitektur wird verwässert.

WARUM SOLLTE ICH DIESES PRINZIP KENNEN / BERÜKSICHTIGEN?

Eine **Root Cause-Analyse** hilft Entwickler:innen und Architekt:innen nicht nur Symptome zu behandeln, sondern langfristig korrekten, wartbaren und verständlichen Code zu schreiben. Dies ist eine der grundlegenden Praktiken des Clean Code-Ansatzes, die genauso für den Bereich der IT-Security gilt. Ziel ist es wartbare und verständliche Sicherheitsmaßnahmen zu implementieren, die effektiv sind. Nur Symptome zu behandeln kann in einer endlosen Rekursion der Nachjustierung enden.

BEISPIELE FÜR DAS URSACHENBEHEBUNGSPRINZIP

Ein Server wird durch viele Denial of Service-Anfragen aus einem



KONTAKT



BLOG



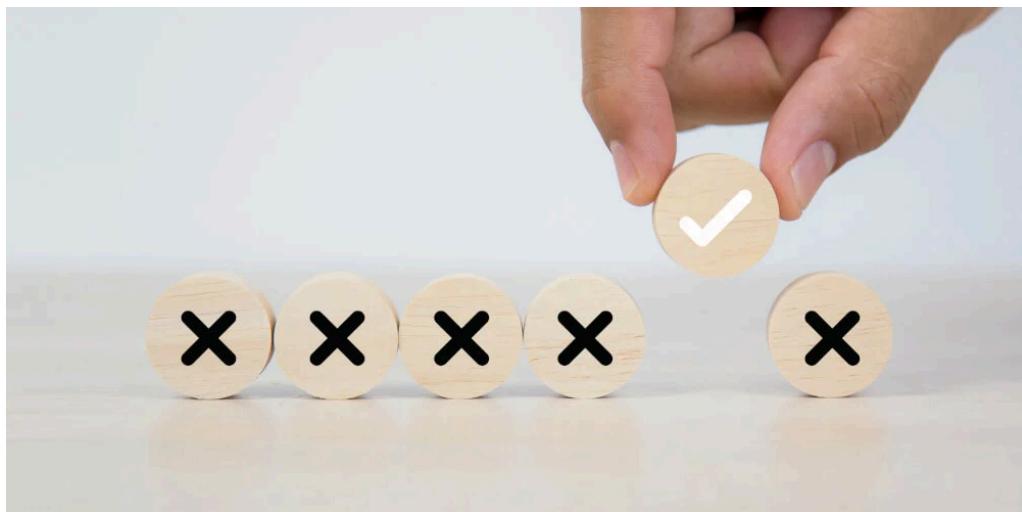
TERMIN



Land gesperrt werden, die Angreifenden unter Einsatz von VPNs jedoch wieder umgehen können.

In einem weiteren Beispiel können in einer Webanwendung Daten erfasst werden, für die der Server keine Input-Validierung vornimmt. Ein:e Angreifer:in kann die Schnittstelle nutzen, um Speicherüberläufe zu provozieren. Als Gegenmaßnahme wird nun anstelle einer serverseitigen Input-Validierung im Client die Eingabelänge für Textfelder begrenzt. Dies behebt nicht den Kern des Problems, so dass weiterhin Angriffe möglich sind.

MINIMIERE PRIVILEGIEN ("LEAST PRIVILEGE")



Berechtigungskonzepte und deren Implementierung sollten immer dem "Least Privilege"-Prinzip folgen. Ein:e Benutzer:in oder ein Prozess sollte nur mit den Berechtigungen ausgestattet sein, die zur Erfüllung der Aufgabe unbedingt erforderlich sind.

WARUM SOLLTE ICH DIESES PRINZIP KENNEN / BERÜCKSICHTIGEN?

Die Autorisierung einer Benutzer:in für eine Funktion oder für Daten erfolgt in der Regel auf Basis zuvor definierter Berechtigungen. Sind die Berechtigungen feingranular umgesetzt und Rollen zugeordnet (**RBAC**), ist es möglich, dedizierte Prüfungen



korrekten Betrieb einer Software erfordert deshalb eine enge Abstimmung zwischen Entwicklung und Betrieb.

BEISPIELE FÜR DAS LEAST PRIVILEGE-PRINZIP

Der Betrieb einer CI/CD-Infrastruktur erfordert einige Entscheidungen in Bezug auf das Least Privilege-Prinzip. Unter anderem sind folgende Fragen in diesem Kontext zu beantworten:

- Auf welche Code-Repositories darf welche:r Entwickler:in zugreifen (lesen/schreibend)?
- Wer darf wohin deployen?
- Welche Berechtigungen haben die verschiedenen Systeme (Versionsverwaltung, Build-System, Artefakt-Repository, ..) untereinander?

Im Kontext des Deployments werden auch häufig Datenbankmigrationen ausgeführt. Diese benötigen in der Regel höhere Berechtigungen als für den normalen Betrieb der Anwendung, da Datenbankschemata modifiziert werden. Das Migrationstool Flyway bietet hier eine Möglichkeit separate User mit höheren Berechtigungen nur für die Migration der Schemata zu nutzen. Die Anwendung selbst baut die Verbindung zur Datenbank danach mit einem eigenen User mit niedrigeren Berechtigungen auf.

Ein weiteres Beispiel für die Anwendung des Least Privilege-Prinzips, bei dem es mitunter zu hitzigen Diskussionen kommen kann, ist die Erteilung von Zugriffsberechtigungen auf die Daten in Produktionssystemen für Entwickler:innen. Aus Entwicklersicht kann ein Zugriff sinnvoll sein, um Fehlersituation zu analysieren. Die Datenschutzperspektive verbietet jedoch in der Regel so einen Zugriff auf Produktionsdaten.

VERMEIDE RISIKEN ("VERMEIDUNGSPRINZIP")



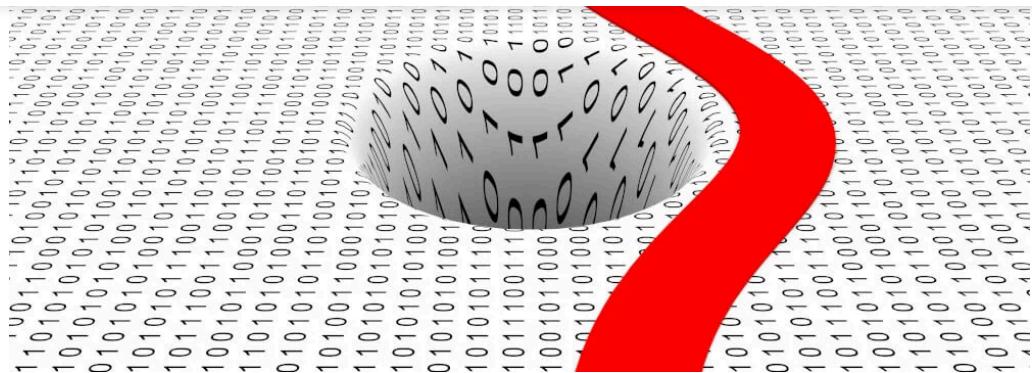
KONTAKT



BLOG



TERMINE



Das Vermeidungsprinzip besagt, dass es unter Abwägung der Chancen und Gefahren sinnvoll sein kann, gewisse Risiken nicht einzugehen und die Angriffsfläche dadurch zu minimieren.

WARUM SOLLTE ICH DIESES PRINZIP KENNEN / BERÜCKSICHTIGEN?

Bei der Architektur eines Software-Systems sind Entscheidungen zu treffen, welche Module miteinander kommunizieren müssen und ob Funktionalitäten selbst implementiert werden oder als Dienst genutzt werden können. Durch die Fokussierung auf die Kernkompetenzen einer Anwendung kann eine Entscheidung sein, sicherheitskritische Funktionalität wie z.B. die Authentifizierung an dafür spezialisierte Systeme (ID-Provider) auszulagern. Ebenso können in der Systemlandschaft sensible Funktionen wie z.B. ein Admin-Interface im Zugriff stärker reglementiert werden.

BEISPIELE FÜR DAS VERMEIDUNGSPRINZIP

Die Kernaufgabe eines Video-Streaming-Dienstes ist das Verwalten und Bereitstellen von Serien und Filmen. Die Identifikation und Authentifizierung von Benutzern sowie die Abrechnung von Abos ist nicht zentraler Mehrwert, den das System schaffen soll. Aus diesem Grund wird der Komplex der Authentifizierung an einen OpenID-Provider delegiert und die Abwicklung der Bezahlung an einen Zahlungsdienstleister wie z.B. Paydirekt.

Eine weitere Vermeidung von Risiken kann der Video-Streaming-Dienst dadurch erreichen, dass das Admin-Interface zur Organisation der Videos, d.h. Upload und Freischaltung von



KONTAKT



BLOG



TERMIN

("MINIMALPRINZIP")

Das Ziel des Minimalprinzips und des Vermeidungsprinzips ist das gleiche: die Minimierung der Angriffsfläche. Bei dem Minimalprinzip geht es im Unterschied zum Vermeidungsprinzip allerdings darum, die Schnittstelle eines Software-Systems so schlank wie möglich zu definieren. Das bedeutet, dass in jedem zusätzlichen Parameter, jeder zusätzlichen Technologie und der Einbindung zusätzlicher Daten die Angriffsfläche vergrößert wird. Dies gilt es zu vermeiden und die Schnittstellen auf das unbedingt erforderliche Maß zu reduzieren.

WARUM SOLLTE ICH DIESES PRINZIP KENNEN / BERÜCKSICHTIGEN?

Schnittstellen zu definieren ist eine häufige Aufgabe von Entwicklern und Architekten. Hierbei konsistent und minimal vorzugehen hat direkten Einfluss auf die Menge der Angriffsvektoren, die einem Angreifer zur Verfügung stehen.

BEISPIELE FÜR DAS MINIMALPRINZIP

Ein Architekt hat die Aufgabe ein System zu entwerfen, das im WWW statische Informationen, die über einen längeren Zeitraum unverändert bleiben werden. Er entscheidet sich gegen den Einsatz eines funktionsreichen CMS-Systems, da dieses unnötig viele Angriffsvektoren eröffnen würde, z.B. über das Interface zur Verwaltung von Inhalten. Stattdessen konzipiert er eine leichtgewichtige Webseite, die statischen Content, d.h. HTML, CSS und Mediendateien ausliefert und keine Möglichkeit zur Verwaltung der Inhalte über ein Web-Interface vorsieht.



KONTAKT



BLOG



TERMIN



in kleinen, für einen bestimmten Zweck spezifischen Einheiten. Damit verhindert er, dass zu viele Informationen ausgeliefert werden, bspw. dass bei einer Abfrage der Bestellhistorie auch Details zu den Zahlungsvorgängen mitgeliefert werden.

**DIE 15 SICHERHEITSPRINZIPIEN
DER SOFTWAREENTWICKLUNG**

CHEAT-SHEET JETZT KOSTENLOS HERUNTERLADEN



viadee®
IT-Unternehmensberatung

UNSERE BLOGSERIE 15 SICHERHEITSPRINZIPIEN DER SOFTWAREENTWICKLUNG

- Teil 1: Prinzip 1 bis 6
- Teil 2: Prinzip 7 bis 10
- Teil 3: Prinzip 11 bis 15

Addon: Laden Sie sich kostenlos unsere Übersicht über die 15 Prinzipien herunter.

Unsere Angebote und Schulungen zu IT-Sicherheit.

Teilen 

ZURÜCK ZUR BLOGÜBERSICHT →

DIESE BEITRÄGE KÖNNTEN SIE EBENFALLS
INTERESSIEREN



Die 15
Sicherheitsprinzipien
der
Softwareentwicklung
(Teil 3)



Die 15
Sicherheitsprinzipien
der
Softwareentwicklung
(Teil 2)



Sicherheit für
Legacy-
Anwendungen in
der Cloud

Keinen Beitrag verpassen – viadee
Blog abonnieren

BLOG
ABONNIEREN

KOMMENTARE

MAARTEN DE KLERK

16.9.2022, 15:43:48

der Beitrag gefällt mir sehr gut!

Reply to *Maarten de Klerk*

Vorname*

Vorname

Nachname*

Nachname

E-Mail*

E-Mail

Website



KONTAKT



BLOG



TERMIN



KOMMENTAR ABSCHICKEN



Andreas Hellmann

Andreas Hellmann ist als IT-Berater und Software-Architekt für die viadee IT-Unternehmensberatung tätig. Sein Schwerpunkt liegt in der Entwicklung von Enterprise-Anwendungen mit Java und Web-Technologien. Zudem engagiert er sich im Kompetenzbereich Security und vermittelt sein Wissen in Schulungen. Andreas Hellmann ist zertifiziert als Certified Information Systems Security Professional (CISSP).

viadee Münster

Anton-Bruchausen-Straße 8, 48147 Münster, Tel: +49 251 77777 0

viadee Köln

Konrad-Adenauer-Ufer 7, 50668 Köln, Tel: +49 221 788807 0

viadee Dortmund

Sebrahweg 7, 44149 Dortmund, Tel.: +49 231 494985 0

www.viadee.de

Service

[Kontakt](#)

[Impressum](#)

[Datenschutzerklärung](#)



KONTAKT



BLOG



TERMIN



viadee



KOMPLEXITÄT BEGREIFEN. LÖSUNGEN SCHAFFEN.