

# IT-Sicherheit

Bedrohungen und Verwundbarkeiten

Version vom 25.09.2024

# Bedrohung – Definition

Eine Bedrohung ist eine **Entität** oder ein **Ereignis** mit dem Potential einen **schädlichen Einfluss** auf ein Informationssystem auszuüben durch

- unautorisierten Zugriff
- Zerstörung
- Offenlegung von Daten
- Veränderung von Daten
- Beeinträchtigung der Verfügbarkeit (denial of service)

[Definition nach (ISC)<sup>2</sup> Common Body of Knowledge]

# Unterscheidung nach Art des Assets

Bedrohungen können unterschieden werden nach dem Typ der betroffenen Ressource:



Gebäude und komplette Standorte

z.B. durch Feuer, Erdbeben, Überflutungen, Terroranschlag



Personen und Organisationen

z.B. durch Social Engineering, Erpressung, Bestechung  
aber auch fehlende Vorgaben, ungeschultes Personal,  
fehlendes Problembewusstsein beim Management



IT-Infrastruktur

z.B. durch Schadsoftware, Stromausfall, Ausfall von Netzwerken

# Unterscheidung nach Quelle

## Natürliche Bedrohungen



- Wetter  
Blitzschlag, Sturm,  
Überschwemmung, Hitzewelle
- Erdbeben, Vulkanismus
- Tiere
- ...

## Menschgemachte Bedrohungen



- Sabotage, Spionage
- Unachtsamkeit
- Diebstahl
- Krieg, Bürgerkrieg
- Terroranschlag

# Schwachstelle

Schwachstellen oder Verwundbarkeiten in einem System können ausgenutzt werden, um die Informationssicherheit zu beeinträchtigen.

## Beispiele für Arten von Schwachstellen



- physische Schwachstellen  
(kaputtes Türschloss, fehlende Einzäunung)



- organisatorische Schwachstellen  
(Chef weist Buchhaltung immer per E-Mail an, Rechnungen zu bezahlen)



- operationelle Schwachstellen  
(ausgefallene Systeme müssen manuell ersetzt werden, überall gleiche Passwörter)

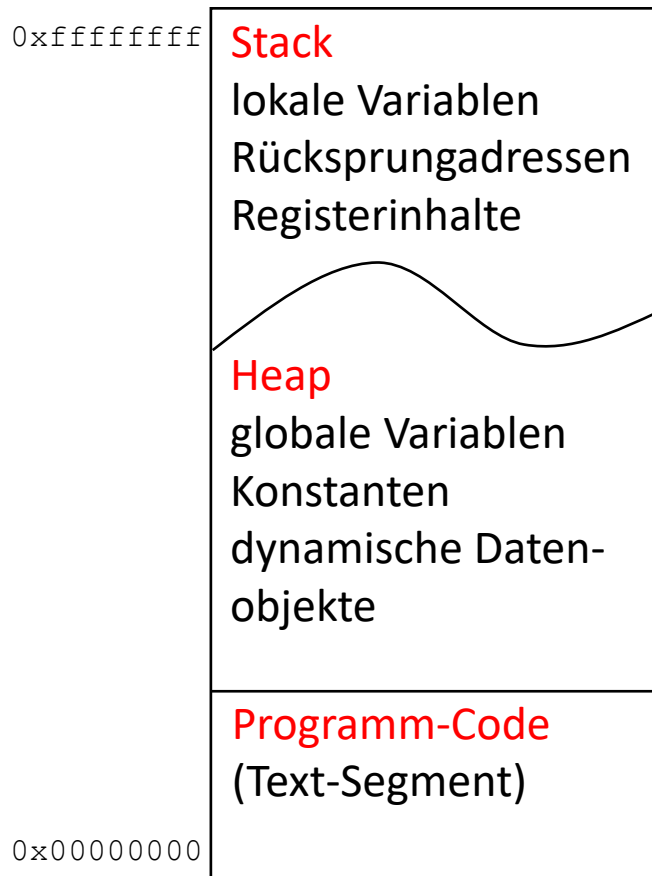


Schwachstellen in Software

# Schwachstellen in Software

- Softwaresysteme stellen i.d.R. sehr **komplexe Systeme** dar.
- Die hohe Komplexität bewirkt, dass **Erstellung, Wartung, Betrieb** von Softwaresystemen **fehlerbehaftete** Prozesse sind.
- Die Umgebung von Softwaresystemen ist ebenfalls hoch komplex und weicht daher immer von den Annahmen des Softwareentwicklers ab.  
Beispiel:  
Die Krypto-Software OpenSSL war ursprünglich als Sammlung von Funktionen für Berechnungen mit großen Zahlen geplant.
- Softwareschwachstellen können unabsichtlich oder absichtlich entstehen.

# Buffer Overflow: Speicheraufbau



- In modernen Betriebssystemen verfügt jeder Prozess über einen eigenen **Virtuellen Adressraum**.
- Dieser wird von Betriebs- und Laufzeitsystem verwaltet.
- Typische Aufteilung: Stack-Segment, Heap-Segment, Text-Segment
- Stack wächst meistens von den hohen zu niedrigen Adressen
- Heap wächst meistens von niedrigen zu hohen Adressen
- Grenze zwischen Heap und Stack oft nicht fest

# Buffer Overflow: Arten

1. Einfaches **Überschreiben von lokalen Variablen** auf dem Stack, um das Programm in einen eigentlich nicht vorgesehenen Zustand zu bringen. (siehe Demo)
2. Überschreiben des Puffers mit Angriffscod und **Überschreiben der Rücksprungsadresse** mit der Adresse dieses Angriffscodes (**Shell Code**).
  - Führt die laufende Funktion ein `return` aus, dann wird an die manipulierte Rücksprungsadresse verzweigt und der Angriffscod ausgeführt.
  - Kennt der Angreifer die Adresse des Puffers nur ungefähr, dann kann er vor den eigentlichen Angriffscod NOP-Assembleranweisungen einfügen. (NOP = NO Operation) → Der Rücksprung muss nur irgendwo in die NOP-Anweisungen erfolgen. (NOP-Slide)
  - Funktioniert nur, wenn Daten auf dem Heap oder Stack ausgeführt werden können. Moderne CPUs/Betriebssysteme verhindern das (**Data Execution Prevention**).



## Buffer Overflow: Arten (cont.)

3. Anlegen einer Liste von Rücksprungadressen, die **auf bereits vorhandene Programmteile** verweisen. (ROP – Return Oriented Programming)
  - Durch geschickte Wahl der Rücksprungadressen werden bereits vorhandene Programmteile so kombiniert, dass ein neues, vom Angreifer gewünschtes Verhalten erzeugt wird.
  - Gegenmaßnahme: Das Programm wird beim Start an zufällige Adressen positioniert (ASLR: Address Space Layout Randomization) → Der Angreifer kann keine Liste von Rücksprungadressen mehr bestimmen.
  - Gegenmaßnahme des Angreifers: Bibliotheken (system32.dll oder libc) liegen oft an festen Adressen → Adressen und Code aus diesen Bibliotheken verwenden.
  - Gegenmaßnahme zu dieser Maßnahme: Auch Systembibliotheken an zufälligen Adressen hinterlegen.
  - Gegenmaßnahme des Angreifers dazu: JIT-ROP, usw. usf.

# Fehlende Prüfung von Eingabedaten

Verlässt sich ein Programm auf Eigenschaften von Eingabedaten ohne diese zu prüfen (**unvalidated input**), dann

- führt dies zu nicht beabsichtigtem Verhalten des Programms, wenn die Eingabedaten die Eigenschaften nicht einhalten.
- kann ein Angreifer u.U. dieses nicht beabsichtigte Verhalten ausnutzen.

Arten von unvalidated input vulnerabilities:

- SQL injection (und andere injection-Schwachstellen)
- Code injection
- Cross Site Scripting
- Cross Site Request Forgery
- ...

# Beispiel: SQL-Injection

Eine Webanwendung liest Nutzernamen und Passwort aus einem Webformular in die Variablen `user` und `password` und prüft sie mit folgender Datenbankabfrage:

```
"SELECT userid FROM users where username='"  
    + user + "'" and password='"  
    + password + "'"
```

Wird eine `userid` von der Datenbank zurückgegeben, dann hat sich der Nutzer erfolgreich angemeldet.

Was passiert bei folgendem "Passwort": `' OR 1=1; --`



# Race Conditions

Verlässt sich ein Programm auf das Auftreten von Ereignissen

- in einer **bestimmten Reihenfolge** oder
- einem **bestimmten zeitlichen Zeitabstand**,

dann kann es zu unbeabsichtigten Verhalten des Programms führen, wenn diese Ereignisse in einer anderen Reihenfolge oder in einem anderen zeitlichen Abstand auftreten, als erwartet.

Besonders anfällig für Race Conditions sind **multi-threaded** Programme und **verteilte Systeme**.

# Race Condition: Beispiel

## Linux Kernel Race Condition in N\_HLDC Driver Lets Local Users Gain Elevated Privileges

SecurityTracker Alert ID: 1037963

SecurityTracker URL: <http://securitytracker.com/id/1037963>

CVE Reference: [CVE-2017-2636](#) *(Links to External Site)*

Date: Mar 7 2017

Impact: [Execution of arbitrary code via local system](#), [Root access via local system](#)

Fix Available: Yes Vendor Confirmed: Yes

Version(s): 4.10.1 and prior

Description: A vulnerability was reported in the Linux kernel. A local user can obtain elevated privileges on the target system.

A local user that can set the HDLC line discipline on the tty device can trigger a race condition and double free memory in the N\_HLDC Linux kernel driver to gain elevated privileges on the target system.

Impact: A local user can obtain elevated privileges on the target system.

<http://www.securitytracker.com/id/1037963>

Wird derselbe Speicher mehrfach freigegeben, so ist das Verhalten nicht definiert!  
Bei den Standardalgorithmen für Speicherverwaltung führt es aber dazu, dass der Angreifer nachfolgende Speicherzuweisungen beeinflussen kann!

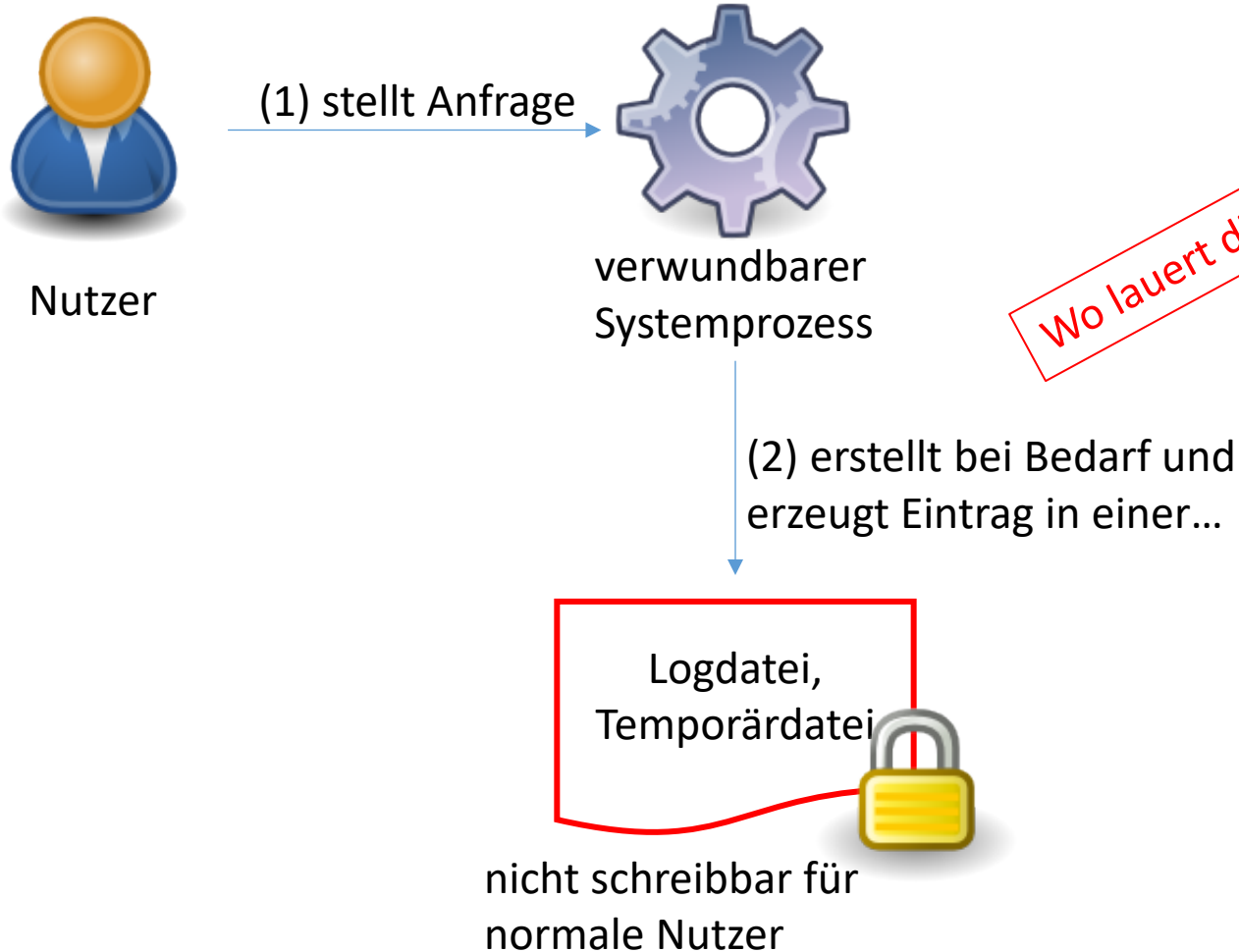
# Unsichere Dateioperationen

Trifft ein Programm Annahmen über Eigentümer, Speicherort oder Attribute einer verwendeten Datei ohne sie zu prüfen bzw. ohne sie so zu **prüfen, dass ein Angreifer sie nicht zwischenzeitlich verändern kann** (vgl. Race Condition), dann kann das zu einer Verwundbarkeit führen.

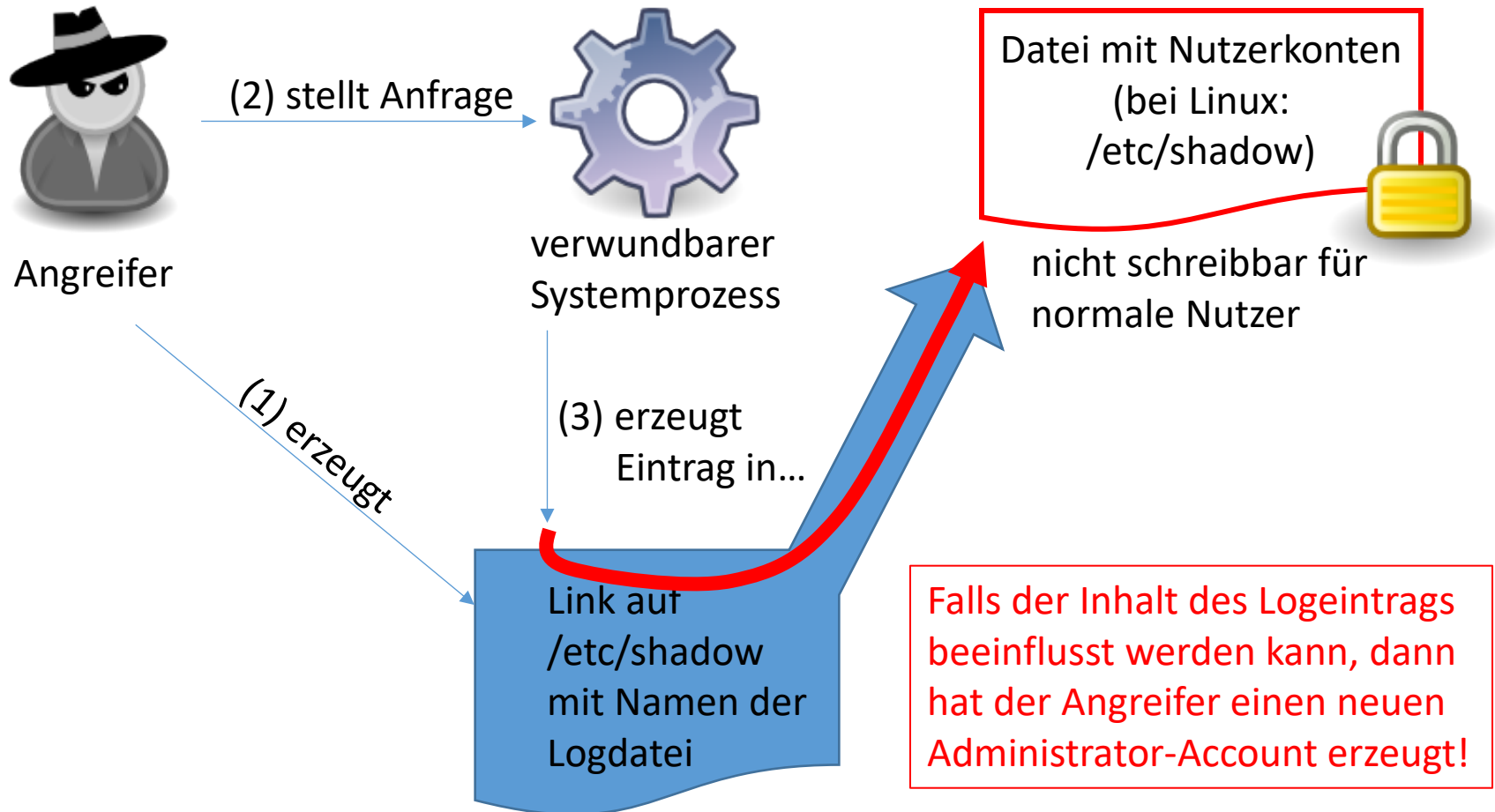
Typische Arten dieser Verwundbarkeiten:

- Lesen/Schreiben von Dateien, die ein anderer Nutzer ebenfalls schreiben kann.
- fehlende Prüfung des Dateityps (Verzeichnis, Link, reguläre Datei, Gerät, ...)
- **fehlende Prüfung des Ergebnisses von Dateioperationen**
- **Annahme, dass Datei lokal ist, wenn sie einen lokalen Dateinamen hat**

# Beispiel unsicherer Dateioperationen



# Beispiel unsicherer Dateioperationen





# Fehlende/unzureichende Zugriffskontrolle

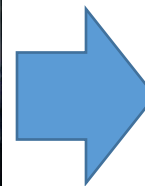
Zugriffskontrolle regelt, **wer** auf **welche Ressourcen** in **welcher Weise** zugreifen darf.

Zugriffskontrolle folgt typischerweise einem bestimmten Schema (dazu in den nächsten Vorlesungen mehr).



# Beispiele fehlerhafter Zugriffskontrolle

- Default-Passwörter, hartkodierte Passwörter, einfach zu erratende Passwörter
- Keine Rechteverwaltung (nur ein Nutzer, der alles darf)
- öffentlich zugängliche Passwörter/Kennungen



<http://news.softpedia.com/news/TV5Monde-Exposes-Passwords-In-Video-Interview-With-Reporter-478158.shtml>

<http://www.france24.com/en/20150409-france-tv5monde-is-group-hacking>

# Typische Schwachstellen/Risiken von Web-Anwendungen

1. Fehlerhafte Zugriffskontrolle
2. Fehlerhafte Anwendung von Kryptographie [ → Veröffentlichung sensibler Daten wie Passwörter, Kreditkartennummern, etc.]
3. Injection (← fehlende Prüfung von Eingabedaten)
4. Unsicheres Design
5. Sicherheitsrelevante Fehlkonfiguration
6. Verwendung von Komponenten mit bekannten Schwachstellen
7. Fehler in Identifikation und Authentisierung
8. Fehler in Software- und Datenintegrität [Fokus: Software-Updates, Fehler in CI/CD-Pipelines]
9. Fehler bei Logging und Monitoring
10. Server-Side Request Forgery

Quelle: OWASP Top 10  
<https://owasp.org>



# Typische Schwachstellen von Software (allgemein)

Eine gute Übersicht bietet das Common Weakness Enumeration (CWE)-Projekt

Rank	ID	Name	Score	2020 Rank Change
[1]	<a href="#">CWE-787</a>	Out-of-bounds Write	65.93	+1
[2]	<a href="#">CWE-79</a>	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	46.84	-1
[3]	<a href="#">CWE-125</a>	Out-of-bounds Read	24.9	+1
[4]	<a href="#">CWE-20</a>	Improper Input Validation	20.47	-1
[5]	<a href="#">CWE-78</a>	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	19.55	+5
[6]	<a href="#">CWE-89</a>	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	19.54	0
[7]	<a href="#">CWE-416</a>	Use After Free	16.83	+1
[8]	<a href="#">CWE-22</a>	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.69	+4
[9]	<a href="#">CWE-352</a>	Cross-Site Request Forgery (CSRF)	14.46	0
[10]	<a href="#">CWE-434</a>	Unrestricted Upload of File with Dangerous Type	8.45	+5
[11]	<a href="#">CWE-306</a>	Missing Authentication for Critical Function	7.93	+13
[12]	<a href="#">CWE-190</a>	Integer Overflow or Wraparound	7.12	-1
[13]	<a href="#">CWE-502</a>	Deserialization of Untrusted Data	6.71	+8
[14]	<a href="#">CWE-287</a>	Improper Authentication	6.58	0
[15]	<a href="#">CWE-476</a>	NULL Pointer Dereference	6.54	-2
[16]	<a href="#">CWE-798</a>	Use of Hard-coded Credentials	6.27	+4
[17]	<a href="#">CWE-119</a>	Improper Restriction of Operations within the Bounds of a Memory Buffer	5.84	-12
[18]	<a href="#">CWE-862</a>	Missing Authorization	5.47	+7
[19]	<a href="#">CWE-276</a>	Incorrect Default Permissions	5.09	+22
[20]	<a href="#">CWE-200</a>	Exposed Functionality	5.09	+22

<https://cwe.mitre.org/top25>

# Organisatorische Schwachstellen

## Führung / Management

- fehlendes Bewusstsein für IT-Sicherheit
- Führen durch falsches Vorbild (Regeln gelten nicht für den Chef)
- keine Ressourcen für IT-Sicherheit bereitstellen
- keine Prüfung der Ergebnisse der IT-Sicherheit
- keine regelmäßigen Audits / Reviews von Status, Änderung, Verbesserung der IT-Sicherheit

## Policies / Richtlinien

- keine IT-Sicherheitsrichtlinien
- zu wenig bekannt im Unternehmen
- unpraktikabel, unpassend für das Unternehmen

## Planung / Betrieb

- keine Risikobewertung
- fehlende Zieldefinition für IT-Sicherheit
- keine Aktionen aus Risikobewertungen