

CWE/OWASP

Dozent: Prof. Dr. Michael Eichberg
Kontakt: michael.eichberg@dhbw.de
Version: 1.1.2

Folien: [HTML] <https://delors.github.io/sec-cwe-owasp/folien.de.rst.html>
[PDF] <https://delors.github.io/sec-cwe-owasp/folien.de.rst.html.pdf>
Fehler melden: <https://github.com/Delors/delors.github.io/issues>

1. Relevante Schwachstellen (CWEs)

CWE-787: Out-of-bounds Write

Beschreibung: Es werden Daten hinter oder vor den Bereich des Puffers geschrieben.

Programmiersprachen:

C /C++

Wahrscheinlichkeit des Missbrauchs:

Hoch

Technische Auswirkungen:

Speichermodifikation; DoS: Crash, Beendigung oder Neustart;
Ausführen von nicht autorisiertem Code oder Befehlen

CWE-787: Out-of-bounds Write - Beispiel 1

```
1 int id_sequence[3];
2
3 /* Populate the id array. */
4
5 id_sequence[0] = 123;
6 id_sequence[1] = 234;
7 id_sequence[2] = 345;
8 id_sequence[3] = 456;
```

CWE-787: Out-of-bounds Write - Beispiel 2

```
1 int returnChunkSize(void *) {
2
3     /* if chunk info is valid, return the size of usable memory,
4      * else, return -1 to indicate an error
5      */
6     ...
7 }
8
9 int main() {
10     ...
11     memcpy(destBuf, srcBuf, (returnChunkSize(destBuf)-1));
12     ...
13 }
```

CWE-787: Out-of-bounds Write - Beispiel 3

```
1 void host_lookup(char *user_supplied_addr){
2     struct hostent *hp;
3     in_addr_t *addr;
4     char hostname[64];
5     in_addr_t inet_addr(const char *cp); // function prototype
6
7     /* routine that ensures user_supplied_addr is in the right format for
8        conversion */
9
10    validate_addr_form(user_supplied_addr);
11    addr = inet_addr(user_supplied_addr);
12    hp = gethostbyaddr( addr, sizeof(struct in_addr), AF_INET);
13    strcpy(hostname, hp->h_name);
14 }
```

Auszug aus der Dokumentation von gethostbyaddr

Return Value

The `gethostbyaddr()` function return the `hostent` structure or a `NULL` pointer if an error occurs.

—<https://linux.die.net/man/3/gethostbyaddr>

CWE-787: Out-of-bounds Write - Beispiel 4

```
1 char *copy_input(char *user_supplied_string){
2     int i, dst_index;
3     char *dst_buf = (char*)malloc(4*sizeof(char) * MAX_SIZE);
4     if ( MAX_SIZE <= strlen(user_supplied_string) ) die("string too long");
5     dst_index = 0;
6     for ( i = 0; i < strlen(user_supplied_string); i++){
7         if( '&'amp;' == user_supplied_string[i] ){
8             dst_buf[dst_index++] = '&';
9             dst_buf[dst_index++] = 'a';
10            dst_buf[dst_index++] = 'm';
11            dst_buf[dst_index++] = 'p';
12            dst_buf[dst_index++] = ';';
13        }
14        else if ( '<' == user_supplied_string[i] ){ /* encode to &lt; */ }
15        else dst_buf[dst_index++] = user_supplied_string[i];
16    }
17    return dst_buf;
18 }
```


CWE-787: Out-of-bounds Write - Beispiel 5

```
1 char* trimTrailingWhitespace(char *strMessage, int length) {
2     char *retMessage;
3     char message[length+1];           // copy input string to a
4     int index;                         // temporary string
5     for (index = 0; index < length; index++) { //
6         message[index] = strMessage[index]; //
7     }                                 //
8     message[index] = '\0';            //
9
10    int len = index-1;                 // trim trailing whitespace
11    while (isspace(message[len])) {    //
12        message[len] = '\0';          //
13        len--;                        //
14    }                                 //
15
16    retMessage = message;
17    return retMessage;                // return trimmed string
18 }
```

Auszug aus der Dokumentation von isspace

If an argument (character) passed to the isspace() function is a white-space character, it returns non-zero integer. If not, it returns 0.

CWE-787: Out-of-bounds Write - Beispiel 6

```
1 int i;
2 unsigned int numWidgets;
3 Widget **WidgetList;
4
5 numWidgets = GetUntrustedSizeValue();
6 if ((numWidgets == 0) || (numWidgets > MAX_NUM_WIDGETS)) {
7     ExitError("Incorrect number of widgets requested!");
8 }
9 WidgetList = (Widget **)malloc(numWidgets * sizeof(Widget *));
10 printf("WidgetList ptr=%p\n", WidgetList);
11 for(i=0; i<numWidgets; i++) {
12     WidgetList[i] = InitializeWidget();
13 }
14 WidgetList[numWidgets] = NULL;
15 showWidgets(WidgetList);
```

CWE-787: Out-of-bounds Write - Mögliche Abhilfemaßnahmen

- Verwendung einer sicheren Programmiersprache (Java, Rust ...)
- Verwendung von Bibliotheken, die sicher(er) sind (z. B. `strncpy` statt `strcpy`)
- Kompilierung mit entsprechenden Flags, die entsprechende Prüfung aktivieren (z. B. `-D_FORTIFY_SOURCE=2`)
- Kompilierung als Position-Independent-Code

Dies löst nicht das Problem, aber es macht es schwerer eine Schwachstelle auszunutzen.

- Statische Analyse Werkzeuge
- Dynamische Analyse Werkzeuge (z. B. *Fuzzing*, *Fault Injection*, ...)

CWE-79: Improper Neutralization of Input During Web Page Generation

Kurzbeschreibung: Nutzereingaben werden nicht oder falsch bereinigt, bevor sie in die Ausgabe eingefügt werden, die als Webseite für andere Benutzer verwendet wird.

Wahrscheinlichkeit des Missbrauchs:
Hoch

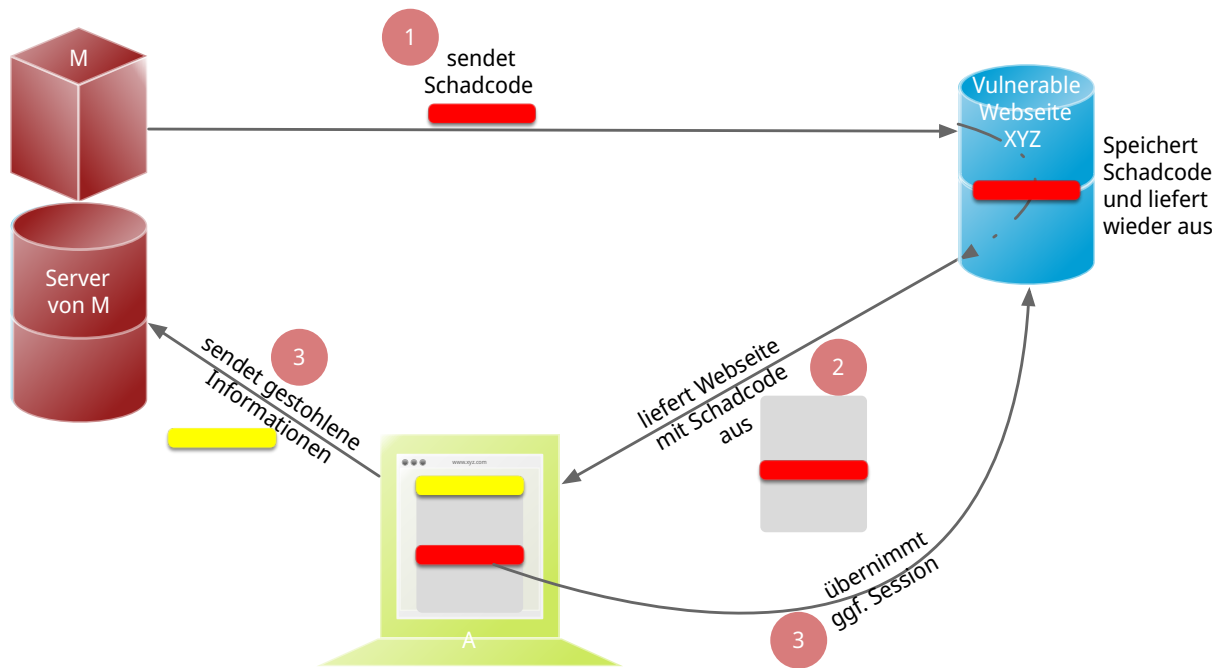
Technische Auswirkungen:
Speichermodifikation; DoS: Crash, Beendigung oder Neustart;
Ausführen von nicht autorisiertem Code oder Befehlen

Betrifft: Zugriffskontrolle, Vertraulichkeit

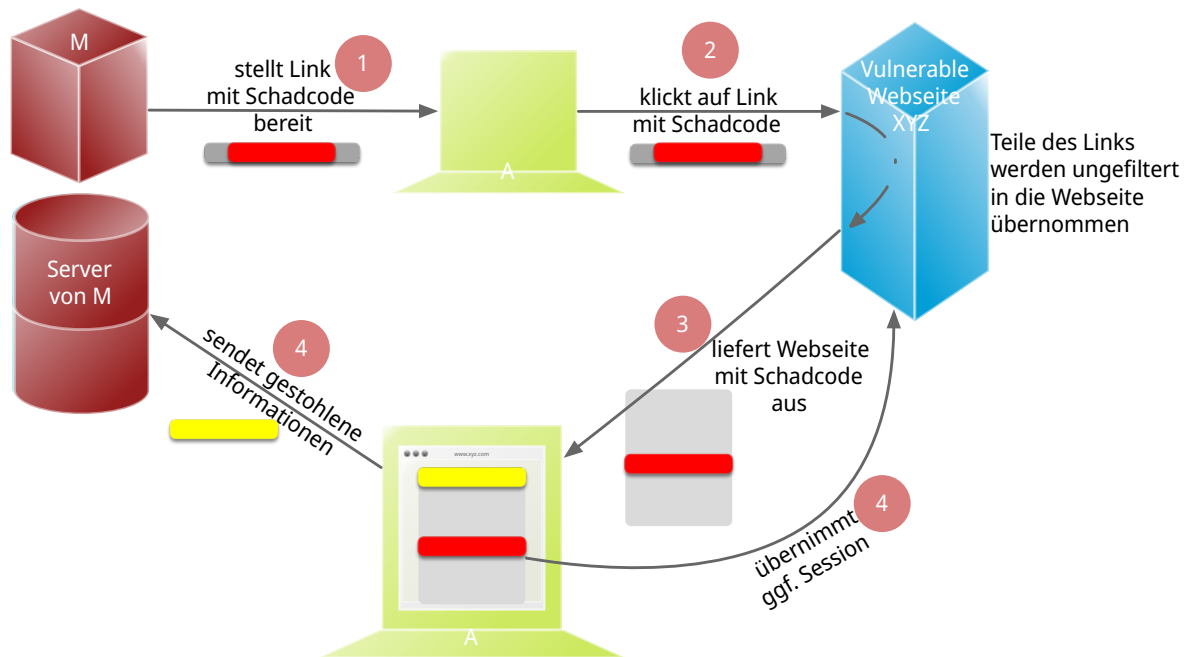
Typen: Stored XSS (Typ 2), Reflected XSS (Typ 1), DOM-based XSS (Typ 0)

Durch eine XSS Lücke werden häufig Informationen abgegriffen (z. B. Session Cookies). Allerdings ist es ggf. auch möglich, dass der Angreifer die Session des Nutzers übernimmt und sich als dieser ausgibt.

Stored XSS (Typ 2)

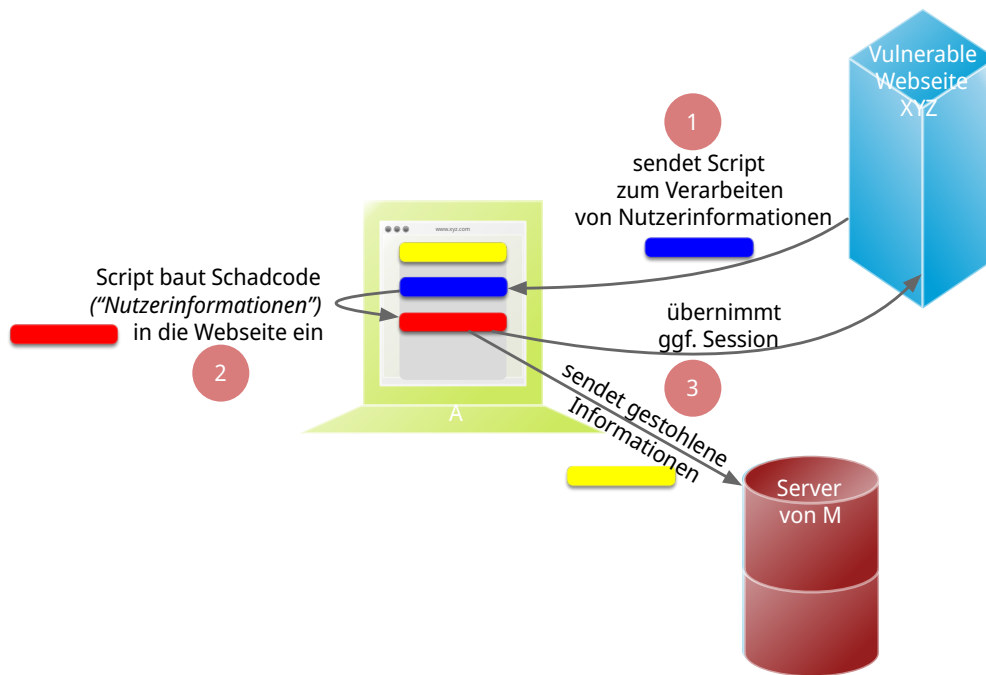


Reflected XSS (Typ 1)



Reflected XSS ist häufig schwerer auszunutzen, da der Angreifer den Nutzer dazu bringen muss, einen Link zu klicken, der den Angriffsvektor enthält. Bei Stored XSS ist dies nicht notwendig, da der Angriffsvektor bereits auf dem Server gespeichert ist.

Dom-based XSS (Typ 0)



Dom-based XSS ist am schwersten Auszunutzen, da der Angreifer den Nutzer dazu bringen muss den Schadcode in die Informationen einzubringen, die von dem Script verarbeitet werden (z. B. durch das Eingeben in ein Formular).

CWE-79: XSS - Beispiel 1 - XSS Typ 1 (Php)

```
1 # Rückgabe einer Willkommensnachricht basierend auf dem
2 # HTTP Get username Parameter
3 $username = $_GET['username'];
4 echo '<div class="header"> Welcome, ' . $username . '</div>';
```

CWE-79: XSS - Beispiel 2 - XSS Typ 2 (JSP)

```
1 <% String eid = request.getParameter("eid");
2   Statement stmt = conn.createStatement();
3   ResultSet rs = stmt.executeQuery("select * from emp where id="+eid);
4   if (rs != null) {
5       rs.next();
6       String name = rs.getString("name");
7   }
8 %>
9
10 Employee Name: <%= name %>
```

CWE-79: XSS - Beispiel 3 - XSS Typ 2 (PHP)

```
1 $username = mysql_real_escape_string($username);
2 $fullName = mysql_real_escape_string($fullName);
3 $query = sprintf('Insert Into users (uname,pwd,fname) Values ("%s","%s","%s")',
4                 $username,
5                 crypt($password),
6                 $fullName) ;
7 mysql_query($query);
8 ...
```

CWE-79: Improper Neutralization of Input During Web Page Generation - Abhilfemaßnahmen und Erkennung

■ **Prüfe jeden Input**

- Prüfung, dass alle auf dem Client durchgeführten Prüfungen auch auf dem Server vorgenommen werden
- Verwendung von geprüften/sicheren APIs
- Verringerung der Angriffsfläche mit dem Ziel möglichst wenig Daten in Cookies etc. zu speichern
- Verwendung von HttpOnly Cookies (d. h. Cookies, die nicht über JavaScript ausgelesen werden können)
- Statische Analyse Werkzeuge
- Beherzigen von Best Practices (**XSS Prevention Cheat Sheet**)
- Setzen der CSP (Content Security Policy)
(Dies löst nicht das Problem, minimiert aber ggf. die Auswirkungen!)

CWE-89: Improper Neutralization of Special Elements used in an SQL Command (*SQL Injection*)

CWE-89: Improper Neutralization of Special Elements used in an SQL Command

Kurzbeschreibung: Ein SQL-Befehl wird ganz oder teilweise unter Verwendung extern beeinflusster Eingaben von einer vorgelagerten Komponente erzeugt. Dabei werden aber spezielle Elemente nicht oder falsch bereinigt, die den beabsichtigten SQL-Befehl verändern könnten, wenn er an eine nachgelagerte Komponente gesendet wird.

Wahrscheinlichkeit des Missbrauchs:

Hoch

Technologie: Datenbanken

Betrifft: Zugriffskontrolle, Vertraulichkeit, Integrität

CWE-89: SQL Injection - Beispiel 1 (MS SQL)

```
1 SELECT ITEM,PRICE
2 FROM PRODUCT
3 WHERE ITEM_CATEGORY='$user_input'
4 ORDER BY PRICE
```

Warnung

MS SQL hat eine eingebaute Funktion, die es erlaubt Shell Befehle auszuführen. Diese Funktion kann auch in einem SQL Statement verwendet werden.

CWE-89: SQL Injection - Beispiel 2 (PHP)

```
1 $id = $_COOKIE["mid"];  
2 mysql_query(  
3     "SELECT MessageID, Subject FROM messages WHERE MessageID = '$id'"  
4 );
```


CWE-89: Improper Neutralization of Special Elements used in an SQL Command - Abhilfemaßnahmen und Erkennung

- Verwendung von geprüften/sicheren APIs
- Verwendung von *Prepared Statements*
- Datenbank nur mit den notwendigen Rechten betreiben
(*Principle of Least Privilege*)
- Sollte es notwendig sein einen dynamischen SQL Befehl zu erstellen, dann sollten geprüfte Escapefunktionen verwendet werden
- Statische Analyse Werkzeuge
- ggf. Application-level Firewall einsetzen

CWE-416: Use After Free

Kurzbeschreibung: Referenzierung von Speicher nach der Freigabe kann dazu führen, dass ein Programm abstürzt, unerwartete Werte verwendet oder Code ausführt.

Wahrscheinlichkeit des Missbrauchs:
Hoch

Programmiersprachen:
C, C++

Betrifft: Verfügbarkeit, Vertraulichkeit, Integrität

CWE-416: Use After Free - Triviales Beispiel

```
1 char* ptr = (char*)malloc (SIZE);
2 if (err) {
3     abrt = 1;
4     free(ptr);
5 }

6 // ... somewhere else in the code
7 //     char* otherPtr = (char*)malloc (SIZE);
8 //     otherPtr* = <HACKER CONTROLLED VALUE>; ...

10 if (abrt) {
11     // Next: use of ptr after free which uses the hacker controlled value
12     logError("operation aborted before commit", ptr);
13 }
```

Hinweis

Ziel ist es im Allgemeinen eine Referenz auf einen interessanten Speicherbereich zu erhalten, der bereits freigegeben wurde und dann den Inhalt dieses Speicherbereichs auszulesen bzw. zu manipulieren, um die nächste Verwendung zu kontrollieren.

CWE-416: Use After Free - Beispiel

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <string.h>
4 #define BUFSIZER1 512
5 int main(int argc, char **argv) {
6     char *buf1R1, *buf2R1, *buf2R2;
7     buf2R1 = (char *) malloc(BUFSIZER1);
8     printf("buf2R1    → %p\n", buf2R1);
9     free(buf2R1);
10    printf("[FREED]    %p\n", buf2R1);
11
12    buf2R2 = (char *) malloc(BUFSIZER1);
13    strncpy(buf2R2, argv[1], BUFSIZER1-1);
14    printf("buf2R2    → %p\n", buf2R2);
15    printf("buf2R2    = %s\n", buf2R2);
16    printf("!!! buf2R1 = %s\n", buf2R1);
17    free(buf2R2);
18 }
```

Fragen:

Wird dieses Program bis zum Ende laufen oder abstürzen?

Welche Ausgabe erzeugt das Programm?

Ist die Ausgabe bei jedem Lauf gleich?

CWE-416: CVE-2006-4997 IP over ATM clip_mkip dereference freed pointer (Linux Kernel)

```
// clip_mkip (clip.c):
198 static void clip_push(struct atm_vcc *vcc, struct sk_buff *skb) {
...
234     memset(ATM_SKB(skb), 0, sizeof(struct atm_skb_data));
235     netif_rx(skb);
236 }
...
// PROBLEMATIC CODE STARTS HERE:
510 clip_push(vcc, skb);
511 PRIV(skb->dev)->stats.rx_packets--;
512 PRIV(skb->dev)->stats.rx_bytes -= len;

// netif_rx (dev.c):
1392 int netif_rx(struct sk_buff *skb) {
...
1428     kfree_skb(skb); //drop skb
1429     return NET_RX_DROP;
```

CWE-416: Use After Free - Abhilfemaßnahmen und Erkennung

- Wahl einer sicheren Programmiersprache (z. B. RUST)
- explizites NULL setzen, nachdem der Speicherbereich freigegeben wurde
- Fuzzing
- Statische Analyse Werkzeuge

Empfohlene Lektüre: [One day short of a full chain: Real world exploit chains explained](#)
(In Teil 1 wird eine UAF Schwachstelle genutzt.)

CWE-78: Improper Neutralization of Special Elements used in an OS Command (*OS Command Injection*)

CWE-78: Improper Neutralization of Special Elements used in an OS Command

Kurzbeschreibung: Alles oder zumindest ein Teil eines Betriebssystembefehls hängt von extern beeinflussten Eingaben ab. Es erfolgt jedoch keine Bereinigung spezieller Elemente, die den beabsichtigten Betriebssystembefehl verändern könnten.

Wahrscheinlichkeit des Missbrauchs:

Hoch

Betrifft: Verfügbarkeit, Vertraulichkeit, Integrität

- Arten:**
1. Ein bestimmtes Program wird ausgeführt und die Nutzerdaten werden als Parameter übergeben.
 2. Die Anwendung bestimmt basierend auf den Nutzerdaten welches Program mit welchen Parametern ausgeführt wird.

CWE-78: Improper Neutralization of Special Elements used in an OS Command - Beispiel (Java)

```
1 ...
2 String btype = request.getParameter("backuptype");
3 String cmd = new String(
4     "cmd.exe /K \"c:\\util\\rmanDB.bat \"
5     +btype+
6     "&&c:\\utl\\cleanup.bat\"")
7
8 System.Runtime.getRuntime().exec(cmd);
9 ...
```

CWE-78: Improper Neutralization of Special Elements used in an OS Command - Abhilfemaßnahmen und Erkennung

- Verwendung von geprüften/sicheren APIs.
- Anwendung bzw. Befehl nur mit den notwendigen Rechten betreiben (*Principle of Least Privilege*) bzw. in einer Sandbox ausführen.
- Statische Analyse Werkzeuge
- Dynamische Analyse in Kombination mit Fuzzing
- Manuelle Code Reviews/Statische Analyse
- ggf. Application-level Firewall einsetzen

CWE-20: Improper Input Validation

Kurzbeschreibung: Empfangene Eingaben oder Daten werden nicht nicht oder falsch validiert in Hinblick darauf, dass die Eingaben die Eigenschaften haben, die für eine sichere und korrekte Verarbeitung der Daten erforderlich sind.

Wahrscheinlichkeit des Missbrauchs:

Hoch

Betrifft: Verfügbarkeit, Vertraulichkeit, Integrität

Anwendungsbereiche:

- Rohdaten - Strings, Zahlen, Parameter, Dateiinhalte, etc.
- Metadaten - Information über die Rohdaten, wie zum Beispiel *Header* oder *Größe*

CWE-20: Improper Input Validation - zu verifizierende Werte und Eigenschaften

Größen:

Z. B. Länge, Häufigkeit, Preis, Rate, Anzahl der Vorgänge, Zeit usw.

Implizite oder abgeleitete Größen:

Z. B. die tatsächliche Größe einer Datei anstelle einer angegebenen Größe.

Indizes*:

Offsets oder Positionen in komplexeren Datenstrukturen.

Schlüssel:

Z. B. von Hashtabellen oder assoziativen Feldern.

Syntaktische Korrektheit:

Übereinstimmung mit der erwarteten Syntax.

Konsistenz:

Z. B. zwischen den Rohdaten und Metadaten oder zwischen Referenzen.

Semantische Korrektheit:

Z. B. Konformität mit domänenspezifischen Regeln, z. B. Geschäftslogik.

Authentizität:

Z. B. von kryptografischen Signaturen.

O'Reilly ist keine SQL Injection



Beobachtung

Ein Name wie O'Reilly stellt ein Problem dar, wenn er in ein SQL Statement eingefügt wird, sollte jedoch von der Anwendung verarbeitet werden können und die Eingabevalidierung passieren.

Hinweis

Die Validierung muss immer in Hinblick auf den Kontext erfolgen.

CWE-20: Improper Input Validation

- Beispiel: Partielle Validierung in C

```
1 #define MAX_DIM 100
2 int m,n, error; /* m,n = board dimensions */
3 board_square_t *board;
4 printf("Please specify the board height: \n");
5 error = scanf("%d", &m);
6 if ( EOF == error ) die("No integer passed!\n");
7 printf("Please specify the board width: \n");
8 error = scanf("%d", &n);
9 if ( EOF == error ) die("No integer passed!\n");
10 if ( m > MAX_DIM || n > MAX_DIM ) die("Value too large!\n");
11
12 board = (board_square_t*) malloc( m * n * sizeof(board_square_t));
13 ...
```

Warnung

Ein vergleichbares Problem ist auch in sicheren Programmiersprachen möglich.

CWE-20: Improper Input Validation - Abhilfemaßnahmen und Erkennung

- (begrenzt) Statische Analyse Werkzeuge
- Manuelle statische Analyse insbesondere in Hinblick auf die zugrundeliegende Semantik
- Dynamische Analyse mit Fuzzing

CWE-125: Out-of-bounds Read

Kurzbeschreibung: Daten vor oder nach einem Puffer werden gelesen.

Wahrscheinlichkeit des Missbrauchs:

Hoch

Programmiersprachen:

C, C++

Betrifft: Vertraulichkeit

Auswirkungen: Umgehung von Schutzmaßnahmen; Lesen von Speicher

Die Ausnutzung dieser Schwachstelle ist häufig schwierig, da nicht immer bekannt ist welche und wie viele Daten gelesen werden können. Es kann allerdings möglich sein Speicheradressen auszulesen. Dies kann ggf. genutzt werden, um Mechanismen wie ASLR zu umgehen.

CWE-125: Out-of-bounds Read - Beispiel: Partielle Validierung in C

```
1 int getValueFromArray(int *array, int len, int index) {
2     int value;
3
4     // check that the array index is less than the maximum length of the array
5     if (index < len) {
6         // get the value at the specified index of the array
7         value = array[index];
8     }
9     // if array index is invalid then output error message
10    // and return value indicating error
11    else {
12        printf("Value is: %d\n", array[index]);
13        value = -1;
14    }
15    return value;
16 }
```

CWE-125: Out-of-bounds Read - Abhilfemaßnahmen und Erkennung

- eine sichere Programmiersprache verwenden
- Fuzzing
- Statische Analyse Werkzeuge welche Kontroll- und Datenflussanalyse durchführen

CWE-22: Improper Limitation of a Pathname to a Restricted Directory (*Path Traversal*)

CWE-22: Improper Limitation of a Pathname to a Restricted Directory

Kurzbeschreibung: Externe Eingaben werden für die Konstruktion eines Pfadnamens verwendet, der eine Datei oder ein Verzeichnis identifizieren soll, das sich unterhalb eines eingeschränkten übergeordneten Verzeichnisses befindet. Eine Bereinigung spezieller Elemente innerhalb des Pfadnamens erfolgt jedoch nicht ordnungsgemäß, was dazu führen kann, dass der Pfadname zu einem Ort außerhalb des eingeschränkten Verzeichnisses aufgelöst wird.

Wahrscheinlichkeit des Missbrauchs:

Hoch

Betrifft: Vertraulichkeit, Integrität, Verfügbarkeit

CWE-22: Path Traversal - Beispiel: fehlende Validierung

PHP:

```
1 <?php
2 $file = $_GET['file'];
3 include("/home/www-data/$file");
4 ?>
```


CWE-22: Path Traversal - Beispiel: partielle Validierung

Perl:

```
1 my $Username = GetUntrustedInput();
2 $Username =~ s/\.\.\\//; # Remove ../
3 my $filename = "/home/user/" . $Username;
4 ReadAndSendFile($filename);
```

Java:

```
1 String path = getInputPath();
2 if (path.startsWith("/safe_dir/")) {
3     File f = new File(path);
4     f.delete()
5 }
```

CWE-22: Path Traversal - Beispiel: verwirrende Python API^[1]

```
1 import os
2 import sys
3 def main():
4     filename = sys.argv[1]
5     path = os.path.join(os.getcwd(),
6                          filename)
7     try:
8         with open(path, 'r') as f:
9             file_data = f.read()
10    except FileNotFoundError as e:
11        print("Error - file not found")
12
13    # do something with file_data
```

Dokumentation os.path.join

*Join one or more path components intelligently. The return value is the concatenation of path and any members of *paths with exactly one directory separator following each non-empty part except the last, meaning that the result will only end in a separator if the last part is empty.*

If a component is an absolute path [...], all previous components are thrown away and joining continues from the absolute path component.

—**Python 3.11.7**

[1] Verwirrende APIs gibt es in praktischen allen Sprachen!

CWE-22: Path Traversal - Abhilfemaßnahmen und Erkennung

- Eingabe vollständig validieren; zum Beispiel über kanonische Pfade
- Sandboxen
- Umgebung härten
- Bei Fehlerausgaben darauf achten, dass keine Informationen über das Dateisystem preisgegeben werden
- den Code mit minimalen Rechten ausführen

CWE-352: Cross-Site Request Forgery (CSRF)

Kurze Beschreibung:

Die Webanwendung prüft nicht bzw. kann nicht prüfen, ob eine Anfrage absichtlich von dem Benutzer gestellt wurde, von dessen Browser sie übermittelt wurde.

D. h. eine CSRF Schwachstelle nutzt das Vertrauen aus, das eine Webseite in den Browser eines Nutzers hat. Bei einem CSRF-Angriff wird ein legitimer Nutzer von einem Angreifer dazu gebracht, ohne sein Wissen eine Anfrage zu übermitteln, die er nicht beabsichtigt hat und auch nicht bemerkt.

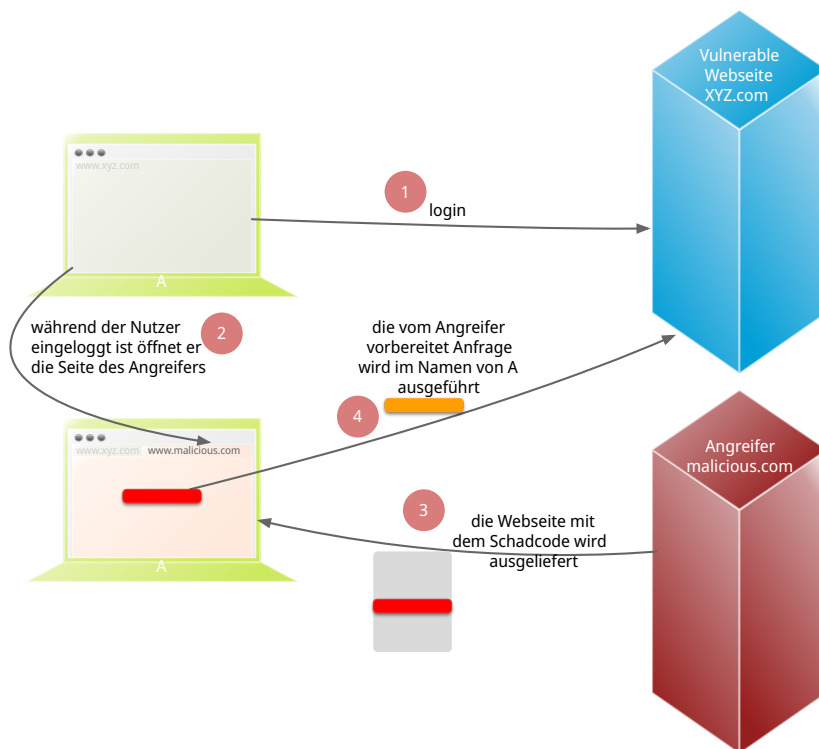
Missbrauchswahrscheinlichkeit:

Mittel

Auswirkung: Hängt von den Nutzerrechten ab

Ausmaß: Vertraulichkeit, Integrität, Verfügbarkeit

CWE-352: Cross-Site Request Forgery (CSRF) - ursprüngliche Form



CWE-352: Cross-Site Request Forgery (CSRF) in 2023

Fiber ist ein von Express inspiriertes Web-Framework, das in Go geschrieben wurde. In der Anwendung wurde eine Cross-Site Request Forgery (CSRF)-Schwachstelle entdeckt, die es einem Angreifer ermöglicht, beliebige Werte zu injizieren und bösartige Anfragen im Namen eines Benutzers zu fälschen. Diese Schwachstelle kann es einem Angreifer ermöglichen, beliebige Werte ohne Authentifizierung einzuschleusen oder verschiedene böswillige Aktionen im Namen eines authentifizierten Benutzers durchzuführen, wodurch die Sicherheit und Integrität der Anwendung gefährdet werden kann. Die Schwachstelle wird durch eine unsachgemäße Validierung und Durchsetzung von CSRF-Tokens innerhalb der Anwendung verursacht.

—**CVE-2023-45128** (übersetzt mit DeepL)

Bemerkung

Identifizierte Schwachstellen: *CWE-20* Improper Input Validation, *CWE-807* Reliance on Untrusted Inputs in a Security Decision, *CWE-565* Reliance on Cookies without Validation and Integrity Checking, **CWE-352** Cross-Site Request Forgery

CWE-352: Cross-Site Request Forgery (CSRF) in 2023

Standardtechniken, die CSRF verhindern *sollen*:

- Same-site Cookies (für Authentifizierung)
- CSRF-Tokens, wenn diese die folgenden Eigenschaften haben:
 - Einmalig pro Nutzersession
 - Geheim
 - nicht vorhersagbar (z. B. eine sehr große, sicher erzeugte Zufallszahl)
- Validierung des Referer-Header
- Custom Request Header, da diese nur vom JavaScript Code gesetzt werden können, der den gleichen Ursprung hat (siehe *Same Origin Policy* (SOP)).

Auch diese Techniken lassen sich ggf. (alle zusammen) aushebeln, **wenn die Anwendung weitere Schwachstellen aufweist**. So gibt/gab es Anwendungen, die Anfragen, die nur über ein POST request gestellt werden sollten, auch bei einem GET akzeptiert haben.

In allen Browsern wird in der Zwischenzeit für Cookies die Same-site Policy angewandt mit dem Wert Lax. Dieser Wert hat zur Folge, dass Cookies nur dann gesendet werden, wenn der Nutzer explizit auf einen Link klickt oder sich innerhalb derselben Seite befindet.

CWE-434: Unrestricted Upload of File with Dangerous Type

Kurze Beschreibung:

Es ist möglich potentiell gefährliche Dateien hochzuladen bzw. zu transferieren, die von der Anwendung automatisch im Kontext der Anwendung verarbeitet werden.

Missbrauchswahrscheinlichkeit:

Mittel

Auswirkung: Bis hin zur Ausführung von beliebigen Befehlen

Ausmaß: Vertraulichkeit, Integrität, Verfügbarkeit

CWE-434: Unrestricted Upload of File with Dangerous Type - Beispiel

HTML:

```
1 <form action="upload_picture.php" method="post" enctype="multipart/form-data">
2     Choose a file to upload:
3     <input type="file" name="filename"/>
4     <br/>
5     <input type="submit" name="submit" value="Submit"/>
6 </form>
```

PHP:

```
1 // Define the target location where the picture being
2 // uploaded is going to be saved.
3 $target = "pictures/" . basename($_FILES['uploadedfile']['name']);
4
5 // Move the uploaded file to the new location.
6 move_uploaded_file($_FILES['uploadedfile']['tmp_name'], $target)
```

CWE-434: Unrestricted Upload of File with Dangerous Type - Abhilfemaßnahmen und Erkennung

- Beim Speichern von Dateien niemals den ursprünglichen Dateinamen verwenden sondern einen vom Server generierten.
- Speicher die Daten nicht im Kontext der Webanwendung sondern außerhalb des Webroots.
- Prüfe die Dateiendung. Prüfe den Inhalt der Datei gegen die Erwartung.
- Ausführen der Webanwendung mit minimalen Rechten.
- Sandbox.

CWE-122: Heap-based Buffer Overflow

Kurze Beschreibung:

Ein Pufferüberlauf, bei dem der Puffer, der überschrieben wird, auf dem Heap alloziiert wurde, was im Allgemeinen bedeutet, dass der Puffer mit einer Routine wie malloc() alloziiert wurde.

Missbrauchswahrscheinlichkeit:

Hoch

Sprachen:

C/C++

Auswirkung:

Bis hin zur Ausführung von beliebigen Befehlen

Ausmaß:

Vertraulichkeit, Integrität, Verfügbarkeit, Zugriffskontrolle

CWE-122: Heap-based Buffer Overflow

„Basisbeispiel“ in C:

```
1 #define BUFSIZE 256
2 int main(int argc, char **argv) {
3     char *buf;
4     buf = (char *)malloc(sizeof(char)*BUFSIZE);
5     strcpy(buf, argv[1]);
6 }
```

CWE-122: Heap-based Buffer Overflow

- Abhilfemaßnahmen und Erkennung

- Verwendung einer sicheren Programmiersprache
- Verwendung von sicheren APIs
- Kompilierung unter Verwendung entsprechender Schutzmechanismen (Position-Independent Executables (PIE), Canaries, ...)
- Härtung der Umgebung (z. B. ASLR)
- Statische Analyse Werkzeuge
- Fuzzing

CWE-502: Deserialization of Untrusted Data

Kurze Beschreibung:

Nicht vertrauenswürdige Daten werden deserialisiert ohne -
je nach Bibliothek notwendige vorhergehende - Prüfung, dass
die Daten die erwarteten Eigenschaften haben.

Missbrauchswahrscheinlichkeit:

Mittel

Sprachen: Java, Ruby, Python, PHP, JavaScript, ...

Ausmaß: Insbesondere: Integrität und Verfügbarkeit (DoS); weitere Effekte
sind vom Kontext abhängig.

Alternative Begriffe:

(Un-)Marshalling, (Un-)Pickling

Bei der Serialisierung werden programminterne Objekte so verpackt, dass die Daten
extern gespeichert und/oder übertragen werden können. Die Deserialisierung kehrt
diesen Prozess um.

CWE-502: Deserialization of Untrusted Data - Beispiel

Java

```
1 File file = new File("object.obj");
2 try ( FileInputStream fin = new FileInputStream(file);
3       ObjectInputStream oin = new ObjectInputStream(fin)
4     ) {
5     javax.swing.JButton button = (javax.swing.JButton) oin.readObject();
6     ...
7 }
```

In diesem Beispiel wird ein Objekt aus einer Datei gelesen und in eine Variable vom Typ `javax.swing.JButton` geschrieben. Der Typ des Objekts wird nicht geprüft. Es ist möglich, dass die Datei ein Objekt enthält, welches vom Typ `javax.swing.JButton` ist, aber nicht die Eigenschaften hat, die ein Button haben sollte. In diesem Fall wird keine Exception geworfen, aber das Objekt kann nicht wie erwartet verwendet werden bzw. es kommt zur Ausführung von beliebigem Code.

CWE-502: Deserialization of Untrusted Data - Beispiel

Python

```
1 class ExampleProtocol(protocol.Protocol):
2
3     def dataReceived(self, data):
4         # ... parse the incoming data and
5         # after receiving headers, call confirmAuth() to authenticate
6
7     def confirmAuth(self, headers):
8         try:
9             token = cPickle.loads(base64.b64decode(headers['AuthToken']))
10            if not check_hmac(token['signature'], token['data'], getSecretKey()):
11                raise AuthFail
12            self.secure_data = token['data']
13        except:
14            raise AuthFail
```

CWE-502: Deserialization of Untrusted Data - Abhilfemaßnahmen und Erkennung

- ggf. Einsatz von Signaturen, um sicherzustellen, dass der serialisierte Code nicht manipuliert wurde
- Serialisiere nur Daten, die auch wirklich serialisiert werden müssen
- Verwendung von sicheren Formaten (z. B. JSON)
- statische Analyse

Empfohlene Lektüre: [Deserialization Vulnerabilities](#)

CWE-918: Server-Side Request Forgery (SSRF)[2]

[2] ≈  *Serverseitige Anfragefälschung*

CWE-918: Server-Side Request Forgery

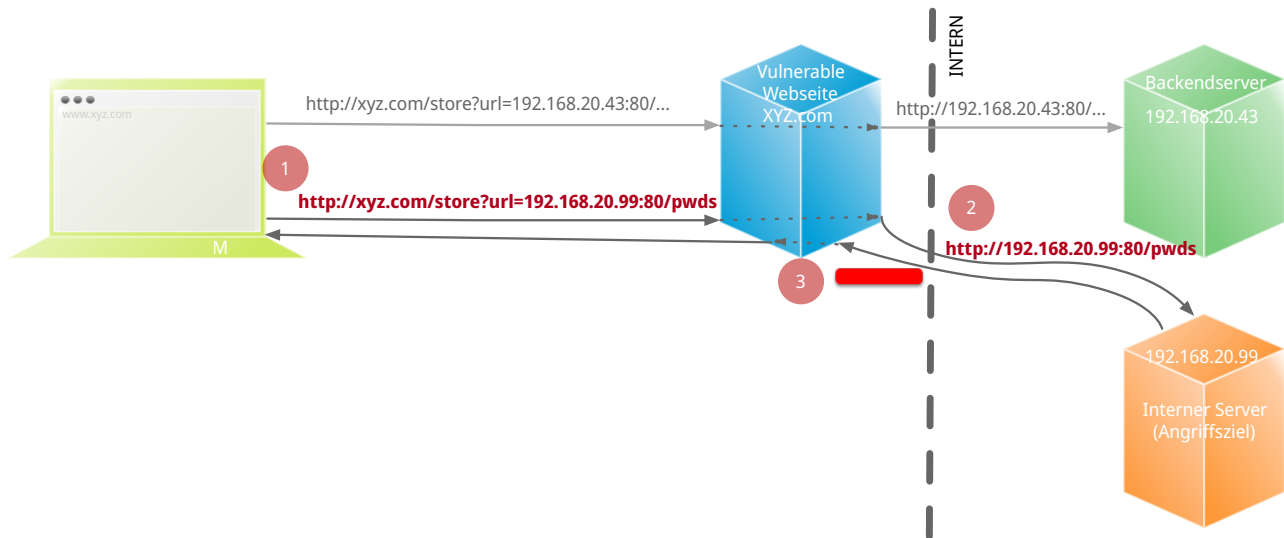
Kurze Beschreibung:

Der Webserver erhält eine URL oder eine ähnliche Anfrage und ruft den Inhalt dieser URL ab, stellt aber nicht sicher, dass die Anfrage an das erwartete Ziel gesendet wird.

Technologien: Webserver

Ausmaß: Vertraulichkeit, Integrität

CWE-918: Server-Side Request Forgery



CWE-918: Server-Side Request Forgery

Beispiel: CVE-2002-1484

Beschreibung: Wenn der DB4Web-Server so konfiguriert ist, dass er ausführliche Debug-Meldungen verwendet, können entfernte Angreifer DB4Web als Proxy verwenden und über eine Anfrage an eine URL, die die Ziel-IP-Adresse und den Port angibt, TCP-Verbindungen zu anderen Systemen (Port-Scan) versuchen, was einen Verbindungsstatus in der resultierenden Fehlermeldung erzeugt.

PoC: <http://127.0.0.1/DB4Web/172.31.93.30:22/foo>

Workaround: Um die Ausnutzung dieses Features zu verhindern, muss die Standardfehlerseite durch eine benutzerdefinierte ersetzt werden. Der Hersteller betrachtet die Funktionalität nicht als Fehler, sondern als Feature für Entwickler.

CWE-918: Server-Side Request Forgery - Beispiel: NodeJS Unicode Handling Fehler [3]

JavaScript:

```
1 var base = "http://orange.tw/sandbox/";  
2 var path = req.query.path;  
3 if (path.indexOf("..") == -1) { // check for no directory traversal  
4     http.get(base + path, callback);  
5 }
```

Beispiel URL (*U+FF2E Full width Latin capital letter N*):

http://orange.tw/sandbox/N N/passwd

≙ http://orange.tw/sandbox/\xFF\x2E\xFF\x2E/passwd

≙ http://orange.tw/sandbox/\x2E\x2E/passwd

≙ http://orange.tw/sandbox/./passwd

[3] Exploiting URL Parsers

CWE-918: Server-Side Request Forgery - Beispiel: URL Parser vs. Abfrage der URL

PHP (> 7.0.13):

```
1 $url = 'http://foo@127.0.0.1 @google.com:11211/'; //  is "just" a space
2 $parsed = parse_url($url);
3 var_dump($parsed[host]); // string(10) "google.com"
4 var_dump($parsed[port]); // int(11211)
5 curl($url);
```

Ergebnis:

`curl` fragt die URL `127.0.0.1:11211` ab.

D. h. `curl` und `php` interpretieren die URL unterschiedlich.

CWE-918: Server-Side Request Forgery

Variante: Blind SSRF

Bei *Blind SSRF*-Schwachstellen werden auch Back-End-HTTP-Anfragen an eine bereitgestellte URL gestellt, die Antwort der Back-End-Anfrage jedoch nicht an die Front-End-Antwort der Anwendung zurückgegeben.


Empfohlene Lektüre: [Blind Server-Side Request Forgery \(SSRF\)](#)

CWE-918: Server-Side Request Forgery

- Abhilfemaßnahmen und Erkennung

- keine (Wieder-)Verwendung der Eingabe URL
- sichere APIs
- statische Analyse (insbesondere Datenflußanalysen)
- Behandlung von Zugriffen von lokalen Maschinen sollte mit der gleichen sorgfalt überprüft werden wie Zugriffe von externen Maschinen; andernfalls können kritische SSRF Angriffe durchgeführt werden
- Firewall/Network Policy, um Zugriff auf interne Systeme zu verhindern

CWE-843: Access of Resource Using Incompatible Type (Type Confusion)

Beschreibung:	Eine Anwendung initialisiert eine Ressource mit einem bestimmten Typ (z. B. Zeiger ( <i>Pointer</i>), Objekt, etc.). Später wird auf die Ressource (Variable) dann mit einem anderen Typ zugegriffen.
Sprachen:	insbesondere (aber nicht ausschließlich) C/C++; im Prinzip in jeder Sprache, die automatische Typkonvertierungen durchführt.
Ausmaß:	Integrität, Verfügbarkeit, Vertraulichkeit

CWE-843: Access of Resource Using Incompatible Type - Beispiel in C

```
1 #define NAME_TYPE 1
2
3 struct MessageBuffer { int msgType; union { char *name; int nameID; }; };
4
5 int main (int argc, char **argv) {
6     struct MessageBuffer buf;
7     char *defaultMessage = "Hello World";
8     buf.msgType = NAME_TYPE;
9     buf.name = defaultMessage; // printf("*buf.name %p", buf.name);
10    buf.nameID = (int)(defaultMessage + 1); // printf("*buf.name %p", buf.name);
11    if (buf.msgType == NAME_TYPE) printf("%s\n", buf.name);
12    else printf("ID %d\n", buf.nameID);
13 }
```

? Frage

Welche Ausgabe erzeugt das Programm?

CWE-843: Access of Resource Using Incompatible Type - Beispiel in Perl

```
1 my $UserPrivilegeArray = ["user", "user", "admin", "user"];
2 my $userID = get_current_user_ID();
3 if ($UserPrivilegeArray eq "user") {
4     print "Regular user!\n";
5 }
6 else {
7     print "Admin!\n";
8 }
9
10 print "\$UserPrivilegeArray = $UserPrivilegeArray\n";
```


CWE-306: Missing Authentication for Critical Function

Beschreibung: Eine Anwendung führt eine kritische Funktion aus, ohne die Identität des Nutzers zu überprüfen. Kritischer Funktionen sind solche, die entweder signifikante Ressourcen verbrauchen oder nur von privilegierten Nutzern ausgeführt werden sollten.

Sprachen: "alle"

CWE-306: Missing Authentication for Critical Function - Abhilfemaßnahmen und Erkennung

- manuelle Code Reviews
- statische Analyse (Binärcode und/oder Quellcode)

White House urges developers to dump C and C++

Biden administration calls for developers to embrace memory-safe programming languages and move away from those that cause buffer overflows and other memory access vulnerabilities.

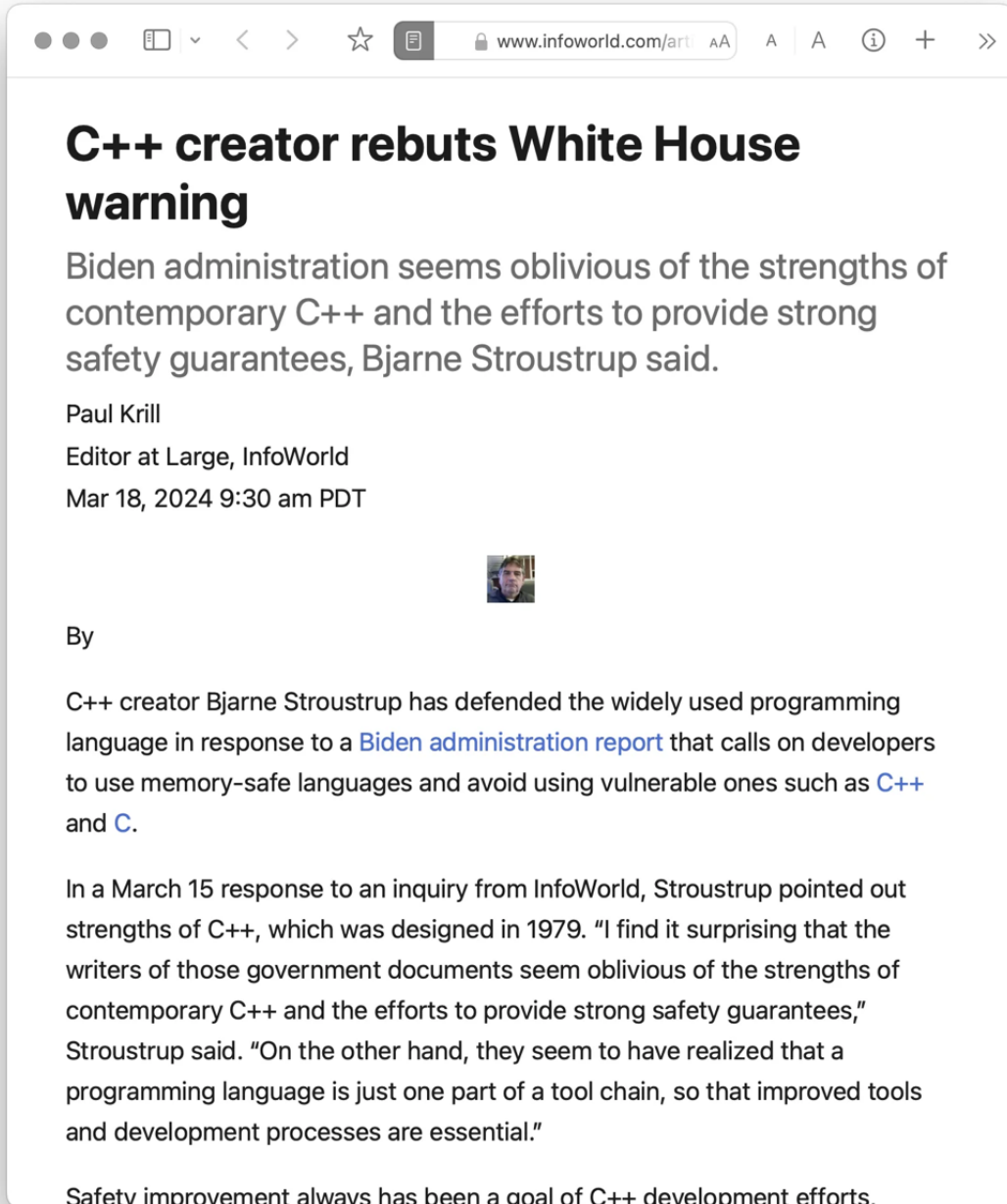


By Grant Gross

InfoWorld | FEB 27, 2024 10:35 AM PST



C++ is missjudged by the White House



2. Open Worldwide Application Security Project (OWASP)

OWASP

- gemeinnützige Stiftung, die sich für die Verbesserung der Sicherheit von Software einsetzt
- 2001 gegründet
- weltweit tätig
- Stellt insbesondere Foren, Dokumente und Werkzeuge bereit
- Dokumente, die bei der Entwicklung sicherer Anwendungen unterstützen:
 - [OWASP Web Security Testing Guide](#)
 - [OWASP Code Review Guide](#)
- Ausgewählte Projekte:
 - [OWASP Top 10 \(die relevantesten Sicherheitsprobleme bei Webanwendungen\)](#)
 - [Cheat Sheets](#)
 - [OWASP Dependency-Track](#)
 - [OWASP Web Security Testing Guide](#)

Übung: Schwachstelle(n) (1)

2.1. malloc verstehen

1. Benenne die Schwachstelle(n) entsprechend der CWEs (ohne ID).
2. Identifiziere die für die Schwachstelle(n) relevanten Zeilen im Code.
3. Gebe - falls möglich - einen Angriffsvektor an.
4. Skizziere mögliche Auswirkung der Schwachstelle(n) (z. B. Verlust der Vertraulichkeit, Integrität oder Verfügbarkeit; Umgehung der Zugriffskontrolle; beliebige Codeausführung, ...)

```
1 #include <stdio.h>
2 #include <string.h>
3 void process(char *str) {
4     char *buffer = malloc(16);
5     strcpy(buffer, str);
6     ...
7     // ... definitively executed in the future: free(buffer);
8 }
9 int main(int argc, char *argv[]) {
10     if (argc < 2) { printf("Usage: %s <string>\n", argv[0]); return 1; }
11     process(argv[1]);
12     return 0;
13 }
```

Übung: Schwachstelle(n) (2)

2.2. REST API

Sie analysieren eine REST API die folgendes Verhalten aufweist, wenn man einem Blog einen Kommentar hinzufügen möchte:

```
1 POST /post/comment HTTP/1.1
2 Host: important-website.com
3 Content-Length: 100
4
5 postId=3&comment=This+<post>+was+helpful.&name=Karl+Gustav
```

Fragt man danach den Webservice nach dem Kommentar, dann erhält man folgendes zurück:

```
1 <div class="comment">
2   <div class="name">Karl Gustav</div>
3   <div class="comment">This <post> was helpful.</div></div>
```

Bewerten Sie die Schwachstelle: CWE Name, problematische Codestelle(n), möglicher Angriffsvektor und mögliche Auswirkung.

Übung: Schwachstelle(n) (3)

2.3. SQL Abfrage mit Java

```
1 String query =  
2     "SELECT account_balance FROM user_data WHERE user_name = "  
3     + request.getParameter("customerName");  
4 try {  
5     Statement statement = connection.createStatement( ... );  
6     ResultSet results = statement.executeQuery( query );  
7 }
```

Bewerten Sie die Schwachstelle: CWE Name, problematische Codestelle(n), möglicher Angriffsvektor und mögliche Auswirkung.

Übung: Schwachstelle(n) (4)

2.4. Verwendung von HTTP Query Parametern

Bemerkung

URL Encoding

%20: Leerzeichen

%22: "

%3C: <

%3E: >

%2F: /

Sie beobachten folgendes Verhalten einer Webseite:

Anfrage

```
1 https://my-website.com/search?  
2 term=This%20is%20a%20%3C%22%3Egift%3C%2F%22%3E
```

Antwort

```
1 <div class="search-result">  
2   <div class="title">This is a <">gift</"></div>  
3 </div>
```

Bewerten Sie die Schwachstelle: CWE Name, problematische Codestelle(n), möglicher Angriffsvektor und mögliche Auswirkung.