Binary Exploitation: Format Strings

Fach: Sicherheit in Verteilten Systemen

Kurs: TIN2021AI

Dozent: Amir Hosh

Studenten: Sandesh Dulal, Philipp Erath, Marc Hummel, Samuel Kraut

Das sind wir:

- **Sandesh Dulal**
  E-Mail: dulals.tin21@student.dhbw-heidenheim.de
  Firma: Nokia
- **Philipp Erath**
  E-Mail: erathp.tin21@student.dhbw-heidenheim.de
  Firma: BS software development GmbH & Co. KG
- **Marc Hummel**
  E-Mail: hummelm.tin21@student.dhbw-heidenheim.de
  Firma: artiso solutions GmbH
- **Samuel Kraut**
  E-Mail: krauts.tin21@student.dhbw-heidenheim.de
  Firma: artiso solutions GmbH

Agenda

- Definition
- Historie
- Theorie
- Verteidigungsmaßnahmen
- Praktische Beispiele

Definition

Definition: Binary Exploitation

- Schwachstellen in Softwareanwendungen, die ausnutzt werden, um unerlaubten Zugriff zu erlangen oder die Anwendung zu manipulieren
- Analyse und Manipulation von ausführbaren binären Dateien
- Finden häufig im Verbund mit dem Stack statt

Quelle: [1] [2]

Binary Exploitation mittels Format String

- Tritt auf, wenn Daten eines Eingabe-Strings von der Anwendung als Befehl ausgewertet werden
- Durch nicht ordnungsgemäßes validieren der Eingaben
- Angreifer kann dadurch:
  - Stack lesen/schreiben
  - Programmfluss verändern
  - Segmentierungsfehler verursachen

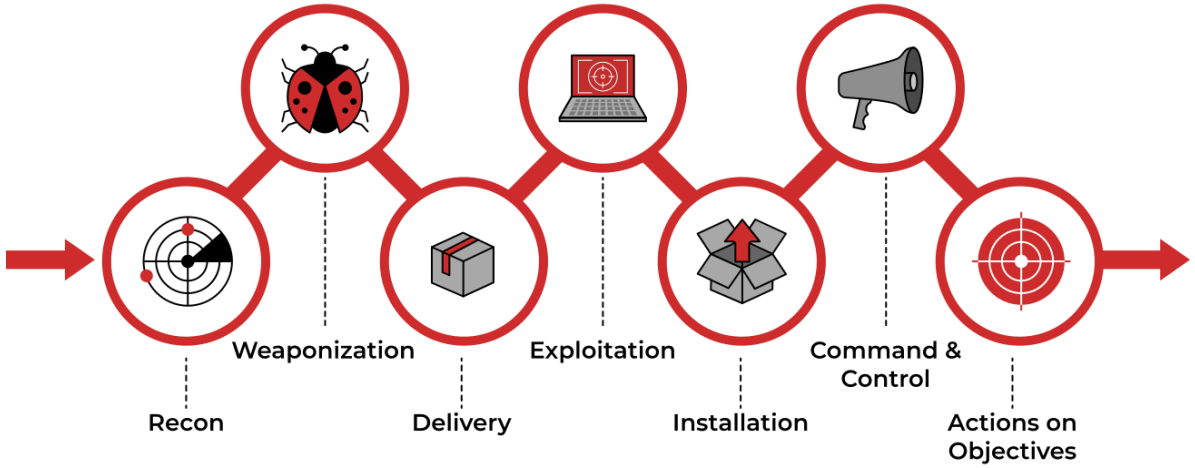
```c
void main()
{
    char format_string[] = "%x";
    printf(format_string);
}

// Output: dff48828
```

Quelle: [2] [3]

Kontext verteilte Systeme

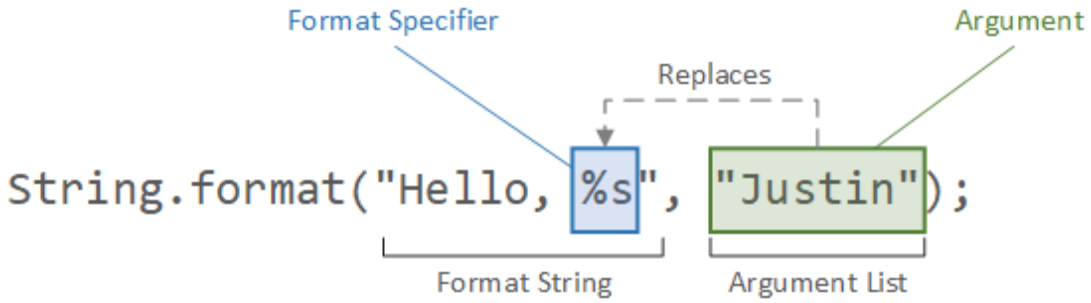
- Vorbereitende Schritte, um Angriff durchzuführen
- Teil der Cyber-Kill-Chain sein
- Python Format Funktion auch ausnutzbar



Definition

Um zu verstehen, wie so eine Exploitation funktioniert, muss man die folgenden Komponenten verstehen:

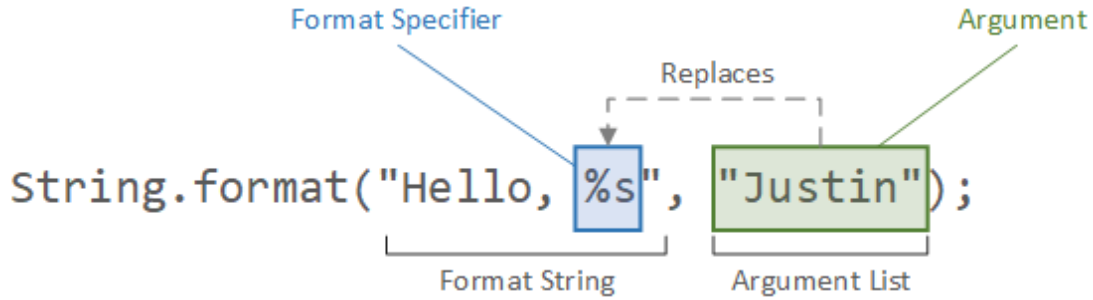
- Format Funktion
- Format String
- Format Specifier



Bildquelle: [2]

Definition: Format Funktion

- Wandelt Variablenwerte des Codes in eine menschenlesbare Zeichenkettendarstellung um
- Um Zeichenketten einfach zu formatieren
- Meistens in Form von Ausgabefunktionen
- Betroffen sind ANSI-C-Konvertierungsfunktionen



Quelle: [2] *Bildquelle:* [2]

Definition: Betroffene Methoden

Format-Funktion	Beschreibung
fprint	Schreibt den printf in eine Datei
printf	Ausgabe einer formatierten Zeichenfolge
sprintf	Ausgeben einer Zeichenfolge
snprintf	Ausgeben eine Zeichenfolge, die die Länge überprüft
vfprintf	Ausgeben Struktur eines va_arg in eine Datei
vprintf	Ausgeben va_arg Struktur nach stdout
vsprintf	Ausgeben va_arg in eine Zeichenfolge
vsnprintf	Ausgeben va_arg in eine Zeichenfolge, die die Länge überprüft

Quelle: [2]

Definition: Betroffene Methoden

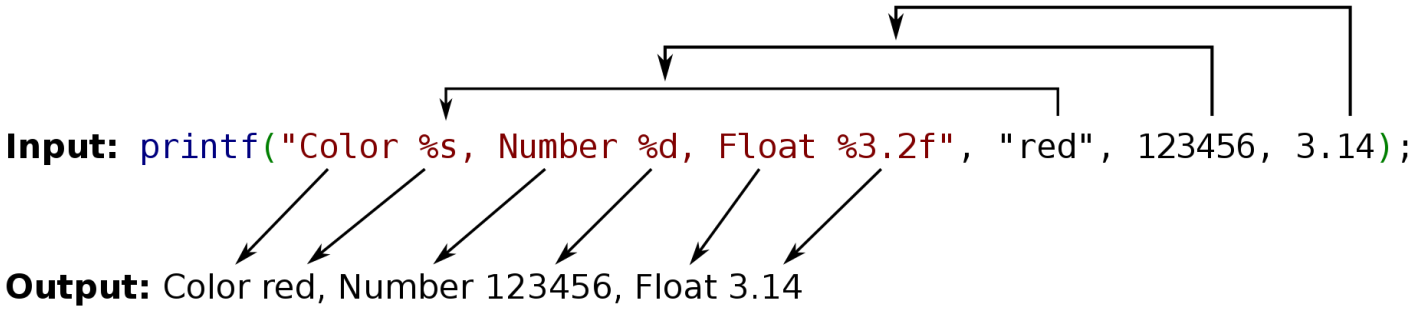
Alle diese Funktionen haben zwei Dinge gemeinsam:

- Format String
- Ausgabe unterschiedlicher Datentypen

Quelle: [3]

Definition: Aufbau printf Methode

- Allgemeine Syntax:
  - printf(format string, values…);
- Genauer:
  - int printf(const char *format, ...);



Bildquelle: [3]

Definition: Aufbau printf Methode

- printf("Color %s, Number %d, Float %3.2f", "red", 123456, 3.14);
- "Color %s, Number %d, Float %3.2f" => Format String
- %s, %d, %3.2f => Format Specifier
- "red", 123456, 3.14 => Argumente/Werte

Definition: Format String

- Ist eine ASCII-Z-Zeichenkette
- Enthält Text und Format Specifier
- In C und in vielen anderen Programmiersprachen verwendet

Quelle: [6]

Definition: Format Specifier

- Platzhalter für Datenelemente
- Formatierung von Argumenten
- Verwendung von % in Zeichenketten
- Ersetzung im Ausgabestrom

Quelle: [3]

Definition: Format Specifier

Parameter	Output	Übergeben als
%p	Darstellung des Werts im Pointer-Stil	Wert
%d	Dezimal	Wert
%c	Zeichen	
%u	Dezimalzahl ohne Vorzeichen	Wert
%x	Hexadezimal	Wert
%s	String	Referenz
%n	Schreibt die Anzahl der Zeichen in einen Zeiger	Referenz

Quelle: [2]

Definition: Problem

- printf() erwartet so viele Parameter wie Format Specifier und greift sich die Argumente in 64-Bit pro Parameter vom Stack.

- Wenn sich nicht genügend Parameter auf dem Stack befinden, werden einfach die nächsten Werte genommen

Quelle: [2]

Gefahren

- Gut gestaltete Eingaben ändern das Verhalten der Format-Funktion
- Kann Denial-of-Service verursachen
- Beliebige Befehle ausführen
- Lesen von und Schreiben in beliebige Speicherplätze

Quelle: [2] [6]

Historie

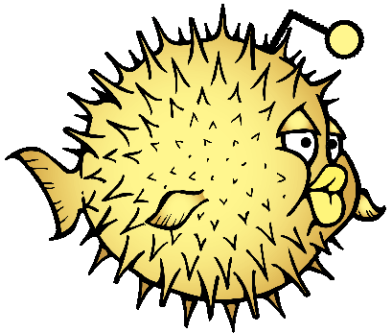
Historie: Entdeckung

- 1989 entdeckt
- 1999 als Angriffsvektor identifiziert
  - von Tymm Twillman während einer Sicherheitsüberprüfung des ProFTPD-Daemons.
- Im Jahr 2000 als Gefahr eingestuft
- Ein Dutzend Exploits (Sicherheitslücke ausgenutzt)
- Wird eingestuft als Programmierfehler

Quelle: [7, 8]

Historie: Reale Angriffe mit Format Strings

- wu-ftpd (2000) (erster Exploit)
  - FTP-Daemon
  - Entwickelt an der Washington University
    - Impact: Remote Root Rechte
- Apache + PHP3
  - Impact: Remote Root Rechte
- OpenBSD fstat
  - Impact: Lokale Root Rechte

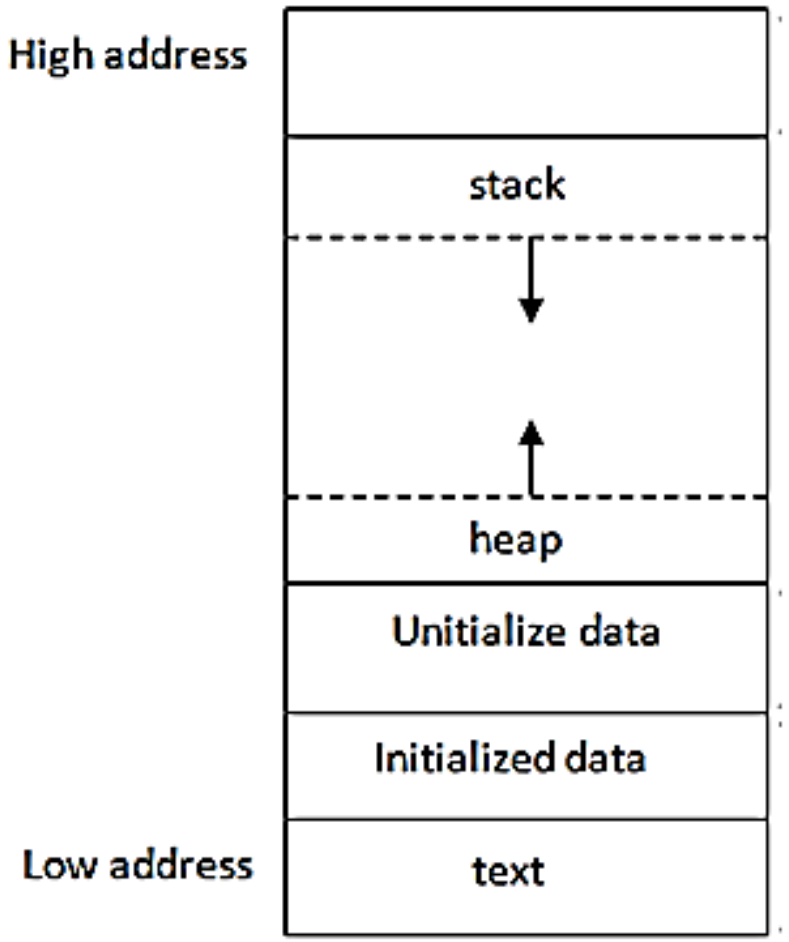


Quelle: [7]

Theorie

Speicherlayout C

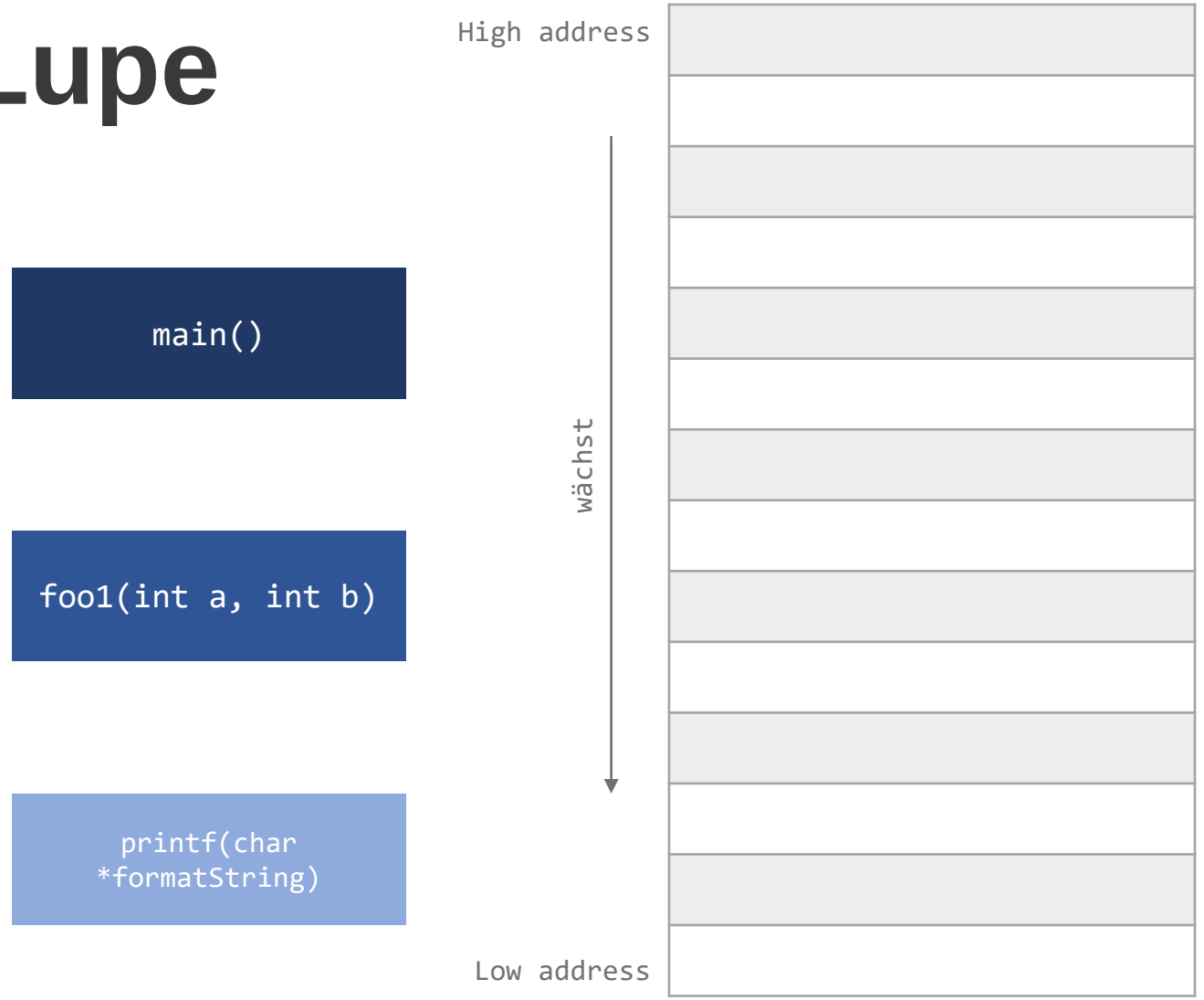
- Jeder Prozess hat sein eigenes getrenntes Speicherlayout
- **Stack**: lokale Variablen (automatisch)
- Heap: Dynamischer Speicher (manuelle Zuweisung)
- Text: Auszuführender Code



Quelle: [9] *Bildquelle:* [4]

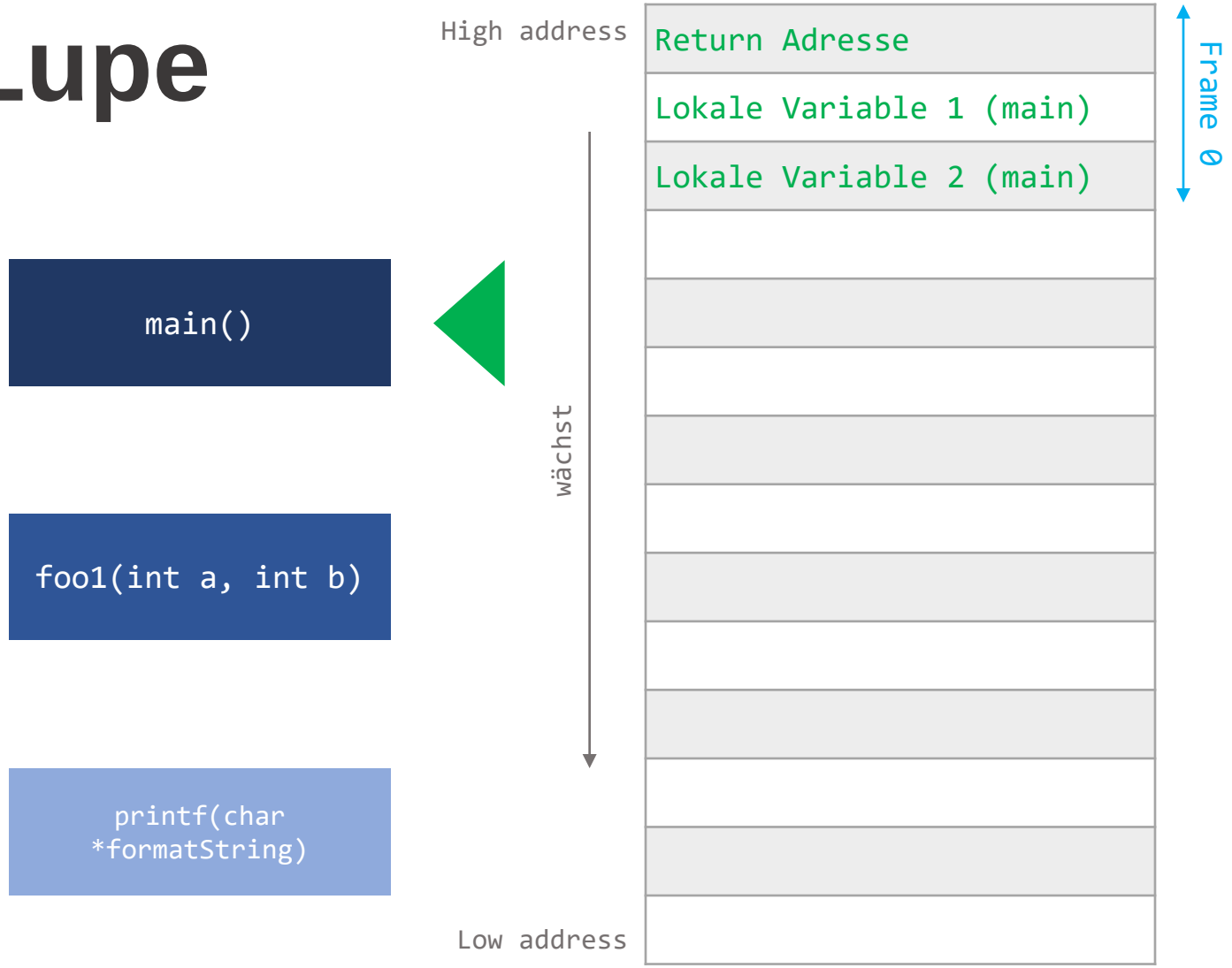
Stack unter der Lupe

- 2 lokale Variablen pro Funktion



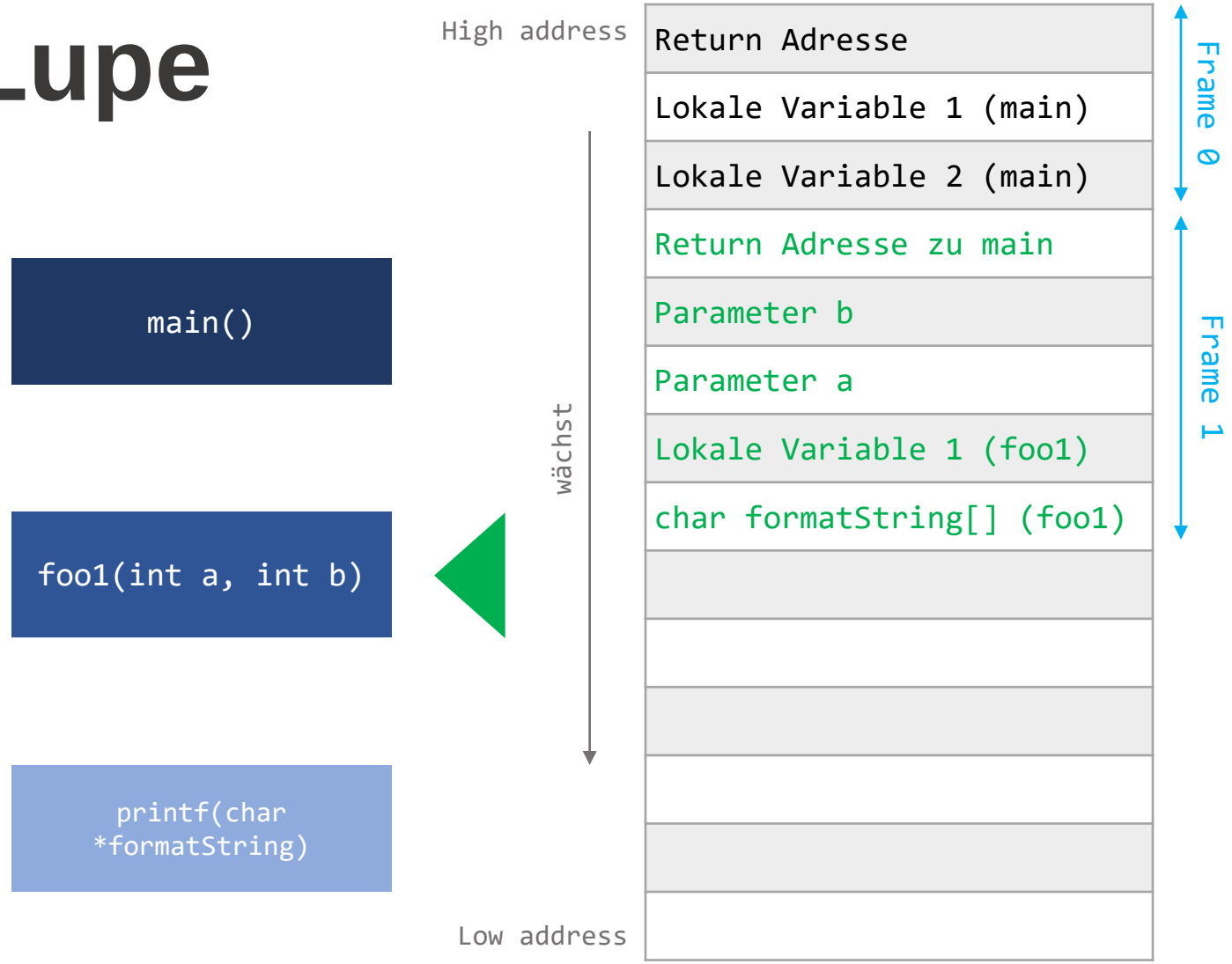
Stack unter der Lupe

- 2 lokale Variablen pro Funktion
- Adresse für Programcounter



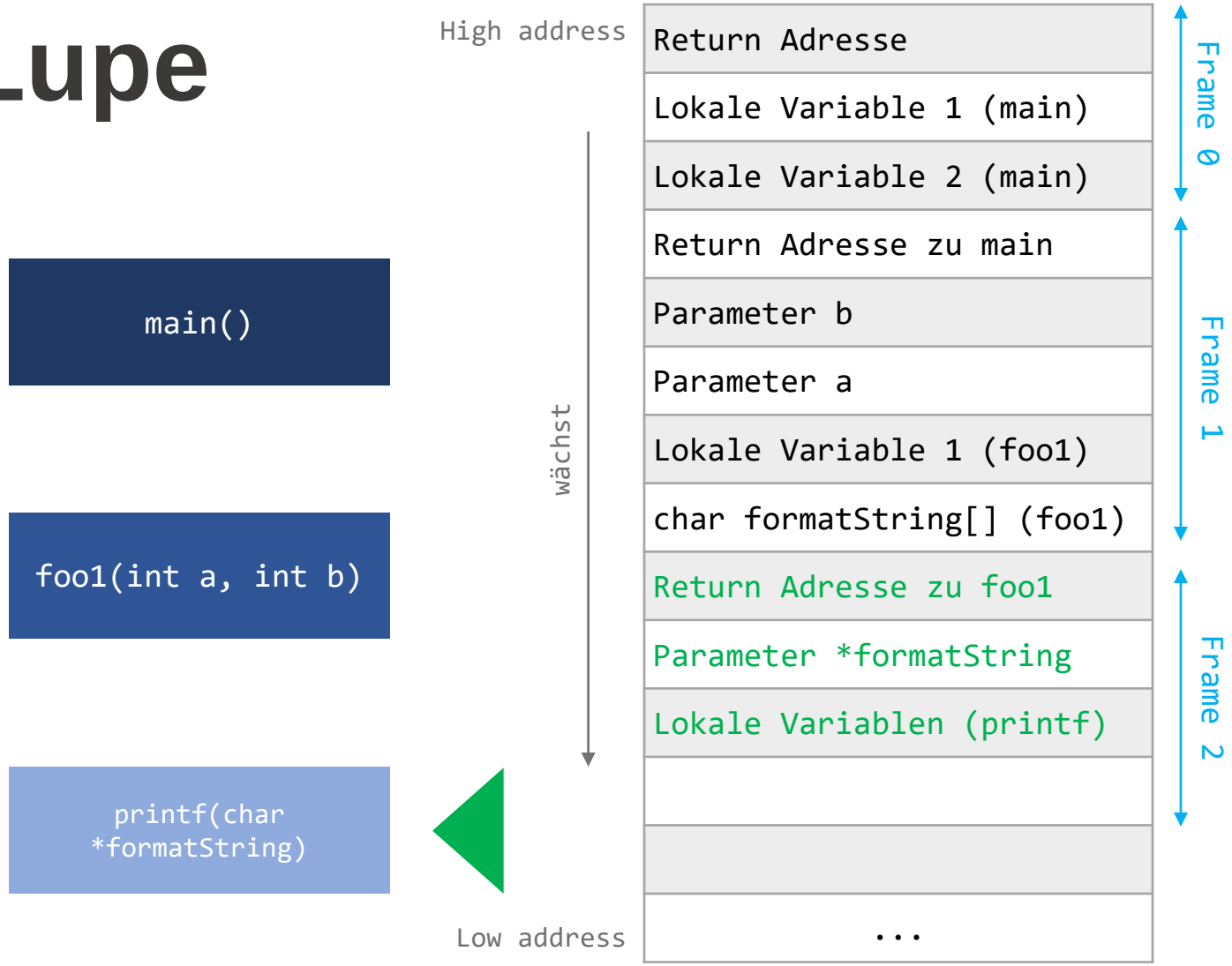
Stack unter der Lupe

- 2 lokale Variablen pro Funktion
- Adresse für Programcounter



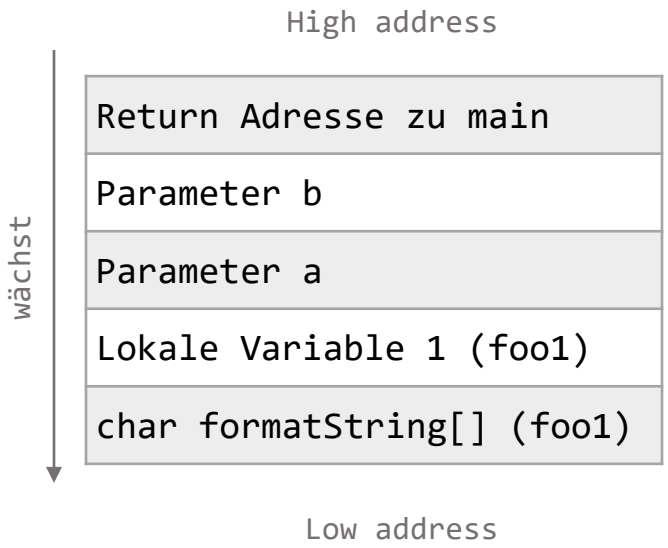
Stack unter der Lupe

- 2 lokale Variablen pro Funktion
- Adresse für Programcounter



Frame

- Datenstruktur
- Informationen über Zustand der Unterprogramme.
- Entspricht Aufruf eines Unterprogramms
- Datenstruktur ist maschinenabhängig.

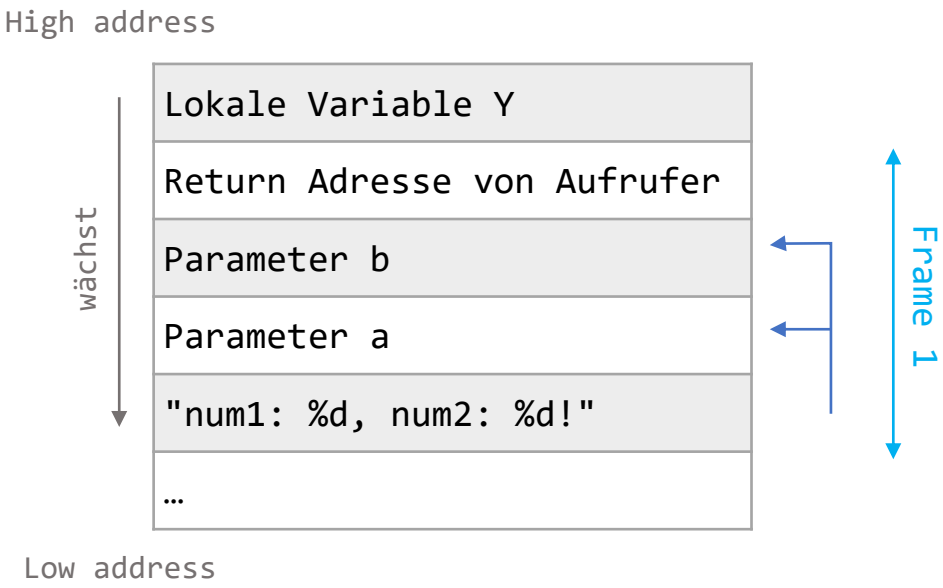


Quelle: [12]

Stack auslesen (1)

Hier: korrekte Anwendung von printf

- `printf("num1: %d, num2: %d!", a, b);`



Stack auslesen (2)

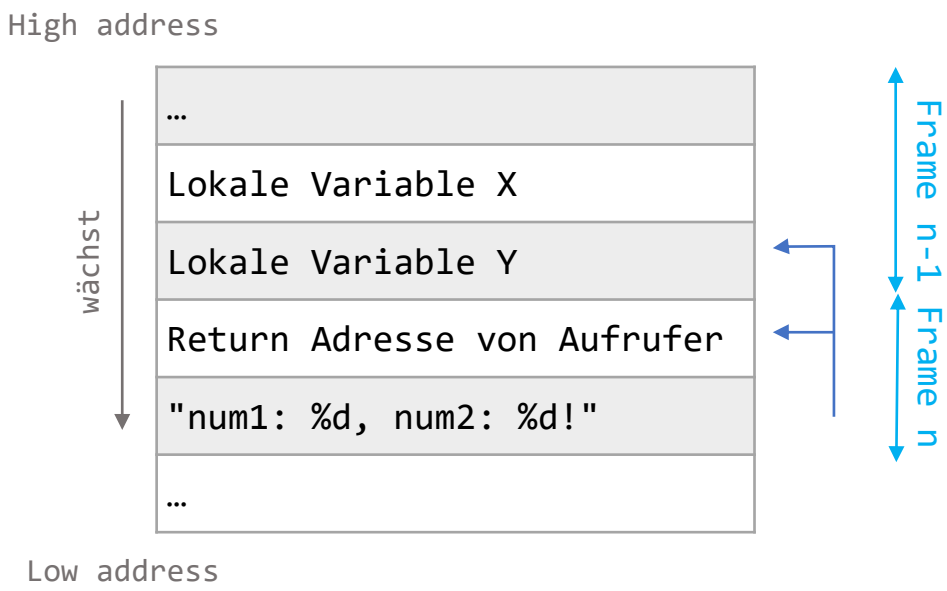
Jetzt: Falsche Anwendung von printf

- `printf("num1: %d, num2: %d!");`

- Problem:
  - Anwendung von Format Identifier OHNE die Übergabe von Argumenten

- Pro Format Identifier => 64 Bit

- Somit kann der eigene Frame verlassen werden ➔ keine boundary checks!



Stack auslesen (3)

- Verschiedene Arten des Auslesens…

Parameter	Output	Übergeben als
%p	Darstellung des Werts im Pointer-Stil	Wert
%d	Dezimal	Wert
%c	Zeichen	Wert
%u	Dezimalzahl ohne Vorzeichen	Wert
%x	Hexadezimal	Wert
%s	String	Referenz

Werte auslesen

- %p, %d, %c, %u und %x lesen den Stack als Werte aus
- Geben **diese Werte** in verschiedenen Formaten (Pointer, Hexadezimal, Dezimal, als Character) aus
- Um ganze 64-Bit auszulesen, ist ein ‚l‘ voranzustellen: %lx
- Um auf das n-te Stack-Elemente zuzugreifen, kann n$ vorangestellt werden: %5$lx

Quelle: [3]

Strings auslesen (1)

- Bei %s hat printf() eine spezielle Routine…
- Erwartet als Argument einen Pointer, der auf den Beginn eines Character-Strings zeigt (endet mit \0)
- %s%s%s%s%s%s%s%s%s%s, um das Programm zu crashen
  - Da es „zufällige“ Werte bekommt, die es als Pointer behandelt

Strings auslesen (2)

- Wie können gezielt Strings ausgelesen werden?
- Je nach Implementierung liegt der Format String selbst auf dem Stack
- Somit kann selbst eine Adresse eingefügt werden, die dann als Argument dient

Adresse	X Stack Pops	%s
\xab\x09\xe3\xff	%Xn	s

Liest String von Adresse 0xffe309ab und gibt diesen aus

Quelle: [3]

Strings auslesen (3)

- Probleme:
  - Adresse des Strings muss bekannt sein (objdump, debugging)
  - Format String muss in lok. Variable abgespeichert sein
  - Strings in C in ASCII-Z-Format: Pointer selbst darf schon kein 0x00 (\0) enthalten, da dies den String terminiert ➔ Untere Adressen sind geschützt

Quelle: [3]

Schreiben einer beliebigen Ganzzahl in den Stack Speicher des Prozesses

Korrekte Verwendung des %n Format Specifier:

```
int i=0;
printf("AAAA%n", &i);
printf("i=%d", i);
```

- Ausgabe:
  AAAA i=4

Vorgehen (1)

1. Stack Anfang ermitteln

2. String erstellen => z.B. "AAAABBBB%6ln"

3. Beim ausführen wird 8 an die erste Stelle im Stack geschrieben

4. String am Anfang um gewünschte Anzahl auffüllen

   Alternativ: Einfügen von Padding um gewünschte Länge zu erreichen

Quelle: [3]

Vorgehen (2)

- Durch mehrfaches Aneinanderreihen von %n werden die Zeiger verschoben

- Beispiel auf einer Little-Endian-32-Bit-Architektur, die "misaligned"-Schreibvorgänge erlaubt ist es möglich, vier aufeinanderfolgende Schreibvorgänge durchzuführen, bei denen jeder Zeiger um eins inkrementiert wird.

- Zwischen den Schreibvorgängen - d. h. zwischen den verschiedenen %n mittels ersten Verfahrens Dummy-Zeichen auszugeben, um das nach dem nächsten Schreiben unberührtem Byte anzupassen.

Quelle: [3]

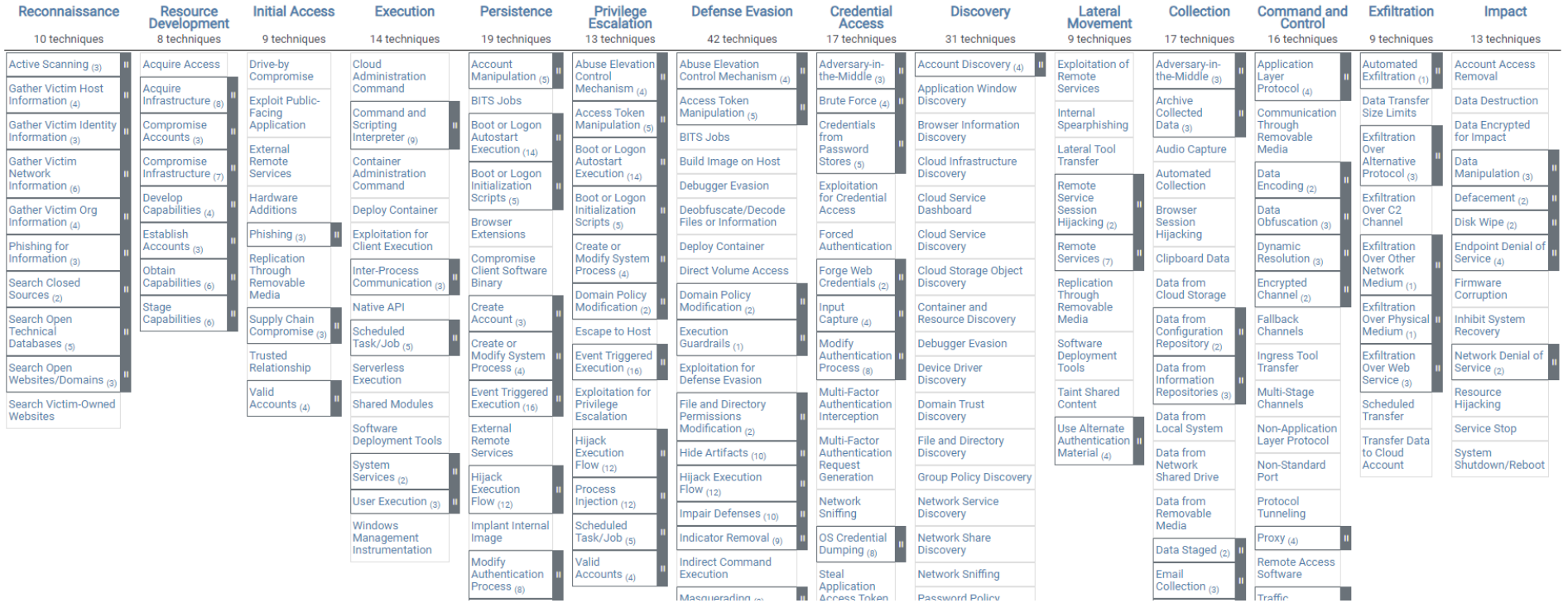
Warum das alles?

- Dient zur 'Privilege Escalation'
- Überschreiben von Flags, um Privilegien zu erlangen
- Überschreiben von Adressen auf:
  - Stack
  - ELF
  - GOT (Global Offset Table)
  - PLT (Program Linking Table)

Quelle: [3]

MITRE ATT&CK

Weltweit zugängliche Wissensbasis über Taktiken und Techniken von Angreifern



Quelle: [5]

Verteidigungsmaßnahmen

Sicherung: Code Analyse

!! Compiler Hinweise nicht ignorieren !!

```
ex1.c: In function ‘main’:
ex1.c:16:1: warning: format not a string literal and
no format arguments [-Wformat-security]
   16 | printf(argv[1]);
      | ^~~~~~
```

Sicherung: Code Analyse

1. Formatierungsfunktionen richtig verwenden
2. Statischen Analyse Tools
    1. C and C++: Flawfinder
    2. PHP: phpsa
    3. Java: Regel FORMAT_STRING_MANIPULATION
3. Manuelle Codeanalyse
4. Unit-Test / System Test
    1. Mögliche Eingaben für Tests auf der OWASP-Website

Quelle: [2]

Sicherung: Neuer Code

- Nie ein String ungeprüft in Formatierungsfunktionen geben

- Ungeprüfte Eingaben mit %s in Formatierungsfunktion geben

```
void main()
{
    char format_string[] = "%x";
    printf("%s",format_string);
}

// Output: %x
```

Quelle: [2]

Fragen

Gibt es offene Fragen / Anmerkungen?

Praktische Beispiele

Quellen (1)

[1] https://dmz.torontomu.ca/wp-content/uploads/2021/03/Binary-Exploitation-201.pdf

[2] https://owasp.org/www-community/attacks/Format_string_attack

[3] http://www.cs.cornell.edu/courses/cs513/2005fa/paper.format-bug-analysis.pdf

[4] https://www.invicti.com/blog/web-security/format-string-vulnerabilities/

[5] https://attack.mitre.org

[6] https://medium.com/swlh/binary-exploitation-format-string-vulnerabilities-70edd501c5be

[7] https://cs155.stanford.edu/papers/formatstring-1.2.pdf

[8] https://en.wikipedia.org/wiki/Uncontrolled_format_string

[9] https://courses.engr.illinois.edu/cs225/sp2020/resources/stack-heap/

[10] https://www.youtube.com/watch?v=Q2sFmqvpBe0

[11] https://www.youtube.com/watch?v=7dMTCdFM2ss

[12] https://en.wikipedia.org/wiki/Call_stack#Structure

[13] https://www.youtube.com/watch?v=y5kcaqKYlqI

Bildquellen (1)

[1] https://medium.com/@WriteupsTHM_HTB_CTF/cyber-kill-chain-tryhackme-7025c0662696

[2] https://dz2cdn1.dzone.com/storage/temp/14967868-string-format.png

[3] https://commons.wikimedia.org/wiki/User:Surachit

[4] https://courses.engr.illinois.edu/cs225/sp2020/resources/stack-heap/