

# 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

# Binary Exploitation mittels Format String

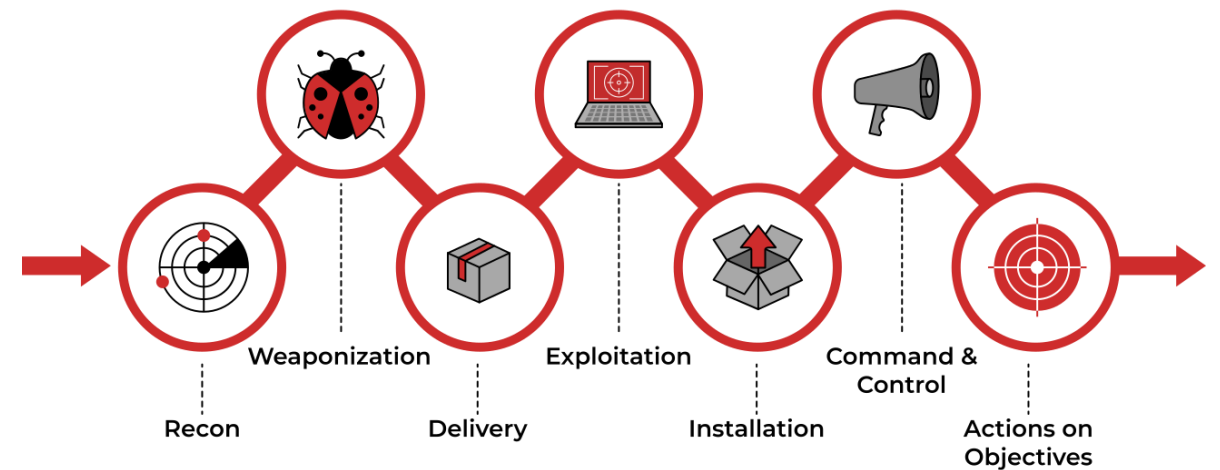
- Tritt auf, wenn Daten eines Eingabe-Strings von der Anwendung als Befehl ausgewertet werden
- Durch nicht ordnungsgemäßes validieren der Eingaben
- Angreifer kann dadurch:
  - Stack lesen/schreiben
  - Programmfluss verändern
  - Segmentierungsfehler verursachen

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

```
// Output: dff48828
```

# Kontext verteilte Systeme

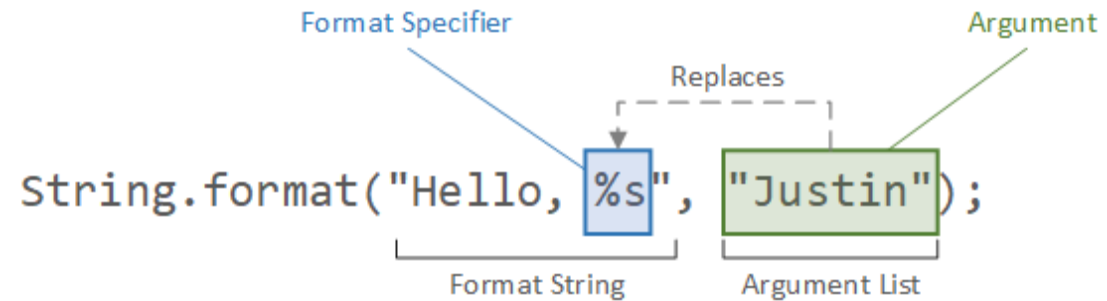
- Vorbereitende Schritte, um Angriff durchzuführen
- Teil der Cyber-Kill-Chain sein
- Python Format Funktion auch ausnutzbar



# Definition

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

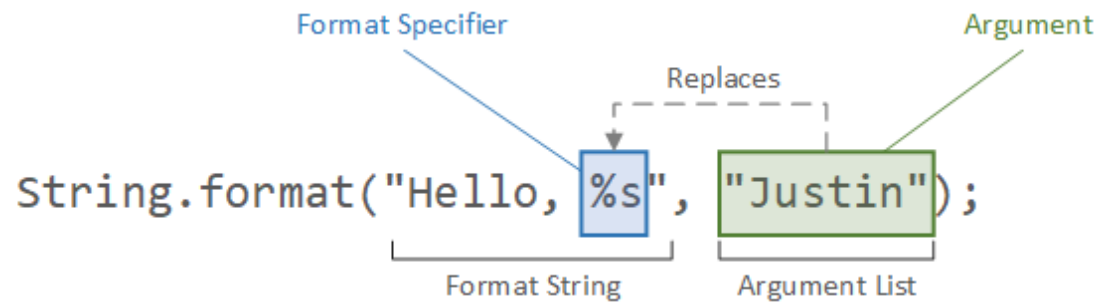
- Format Funktion
- Format String
- Format Specifier





# 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



# 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

# Definition: Betroffene Methoden

Alle diese Funktionen haben zwei Dinge gemeinsam:

- Format String
- Ausgabe unterschiedlicher Datentypen

# Definition: Aufbau printf Methode

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

The diagram illustrates the mapping between the format string and the arguments in a printf statement. Arrows point from each format specifier in the input string to its corresponding value in the output string. Additionally, arrows from the opening and closing parentheses of the printf function point to the first and last arguments, respectively.

**Input:** `printf("Color %s, Number %d, Float %3.2f", "red", 123456, 3.14);`

**Output:** Color red, Number 123456, Float 3.14

# 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

# Definition: Format Specifier

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

# 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



# 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

# 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

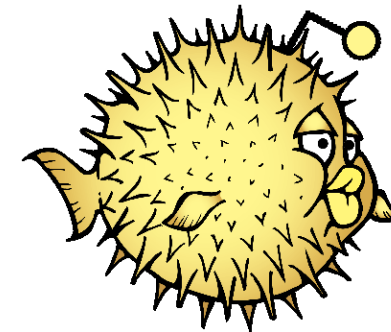
# Historie

# 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

# Historie: Reale Angriffe mit Format Strings

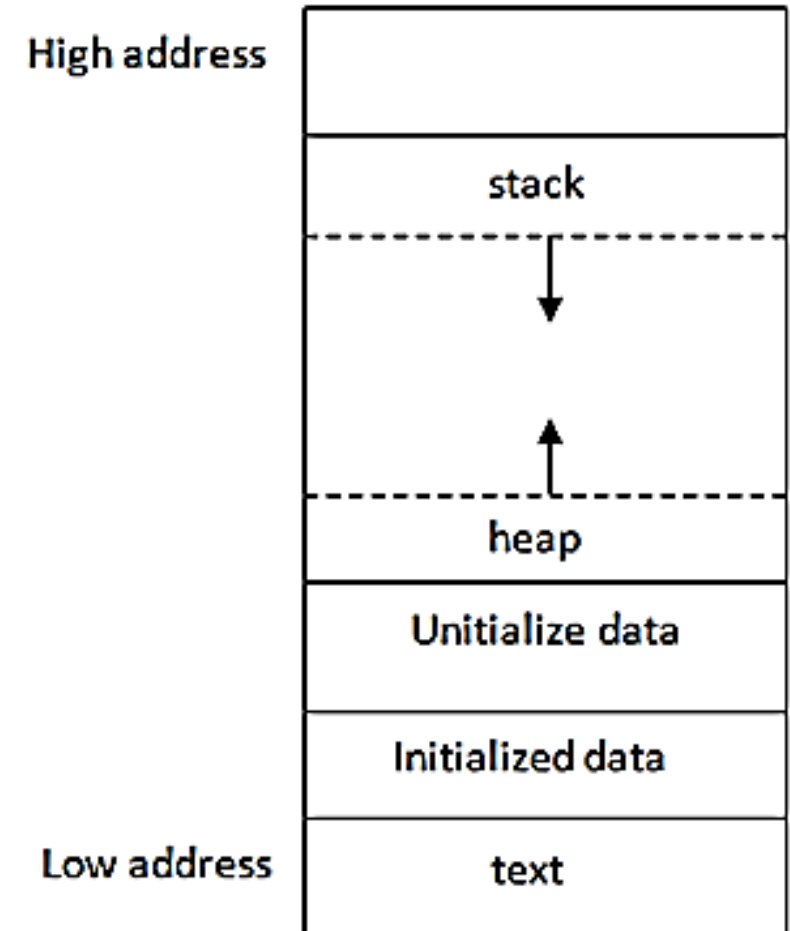
- wu-ftpd (2000) (erster Exploit)
  - FTP-Daemon
  - Entwickelt an der Washington University
    - Impact: Remote Root Rechte
- Apache + PHP3
  - Impact: Remote Root Rechte
- OpenBSD fstat
  - Impact: Lokale Root Rechte



# Theorie

# Speicherlayout C

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



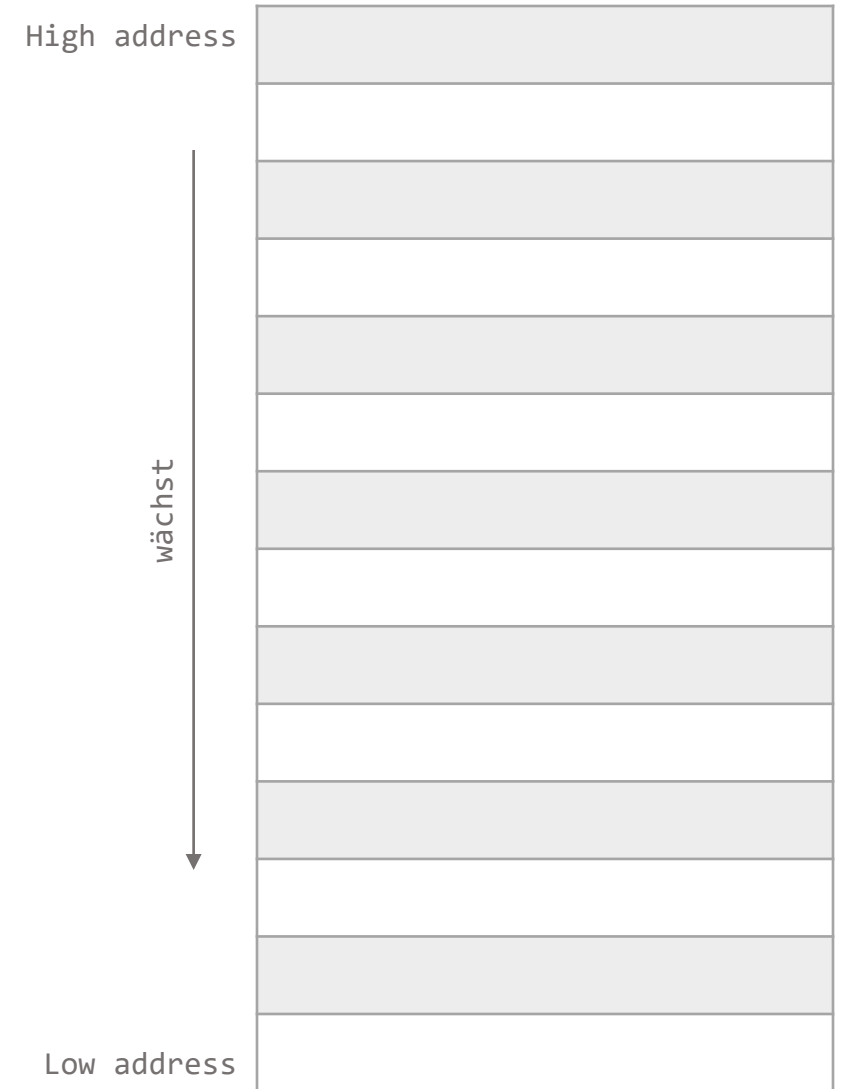
# Stack unter der Lupe

- 2 lokale Variablen pro Funktion

`main()`

`foo1(int a, int b)`

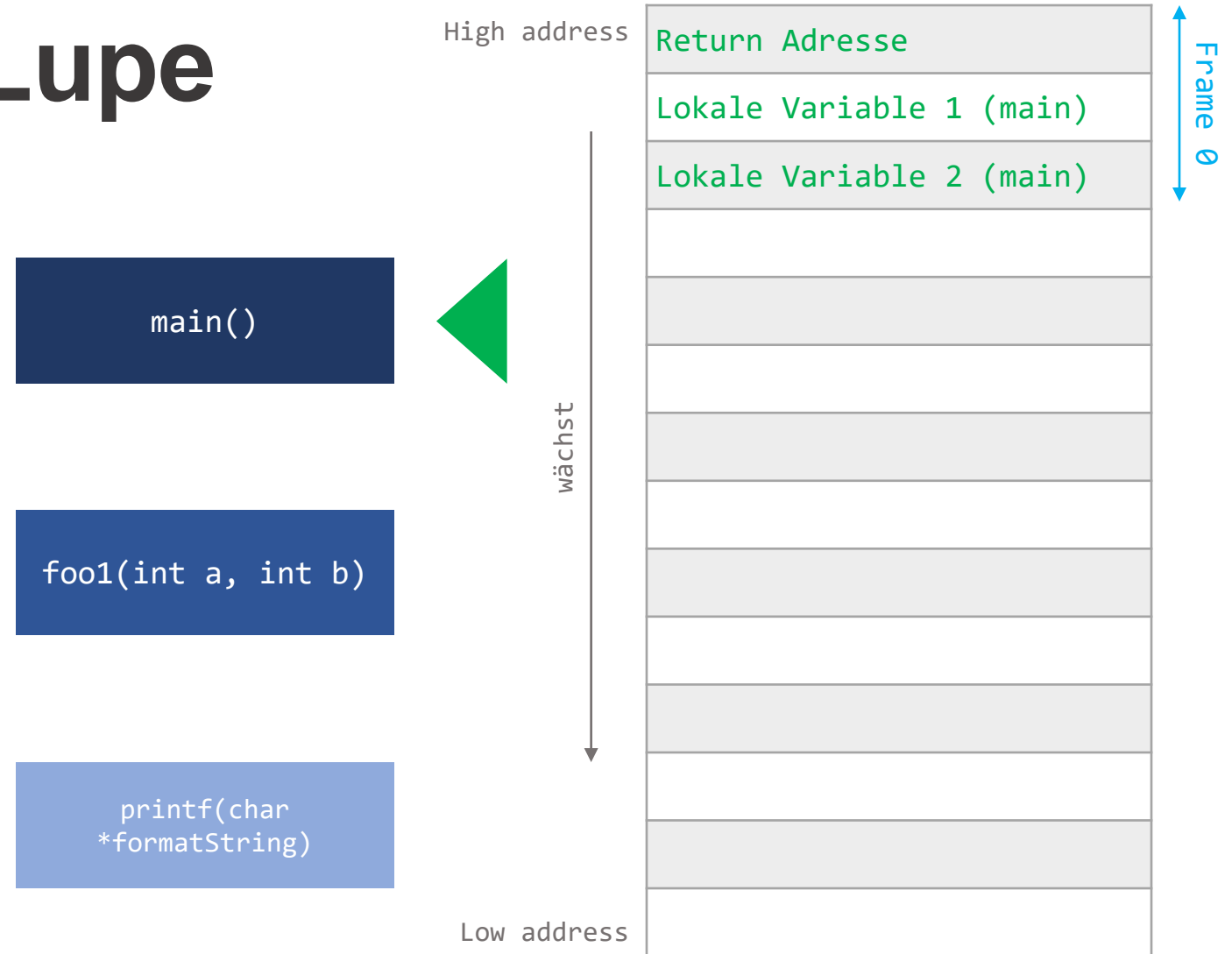
`printf(char  
*formatString)`





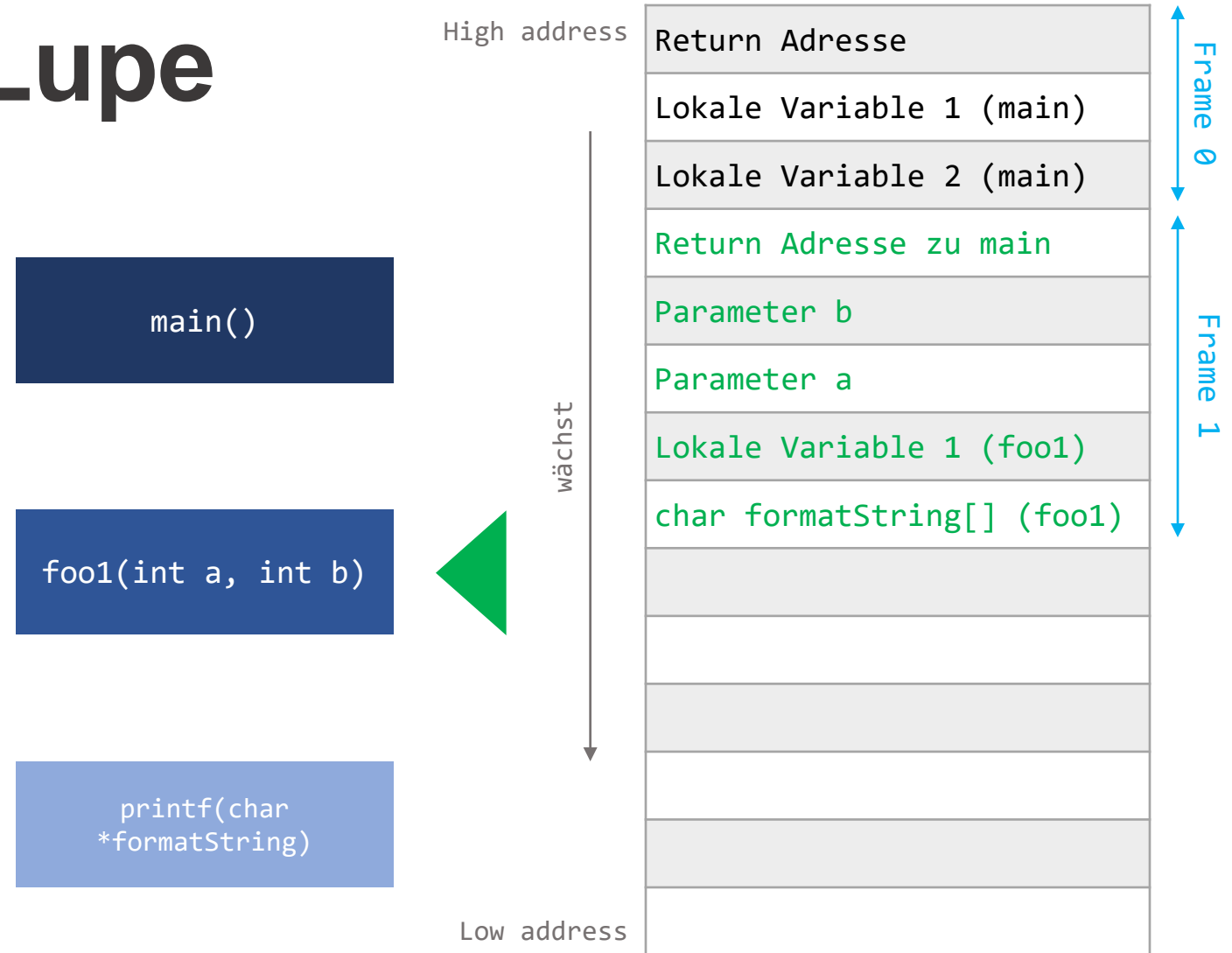
# Stack unter der Lupe

- 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

main()

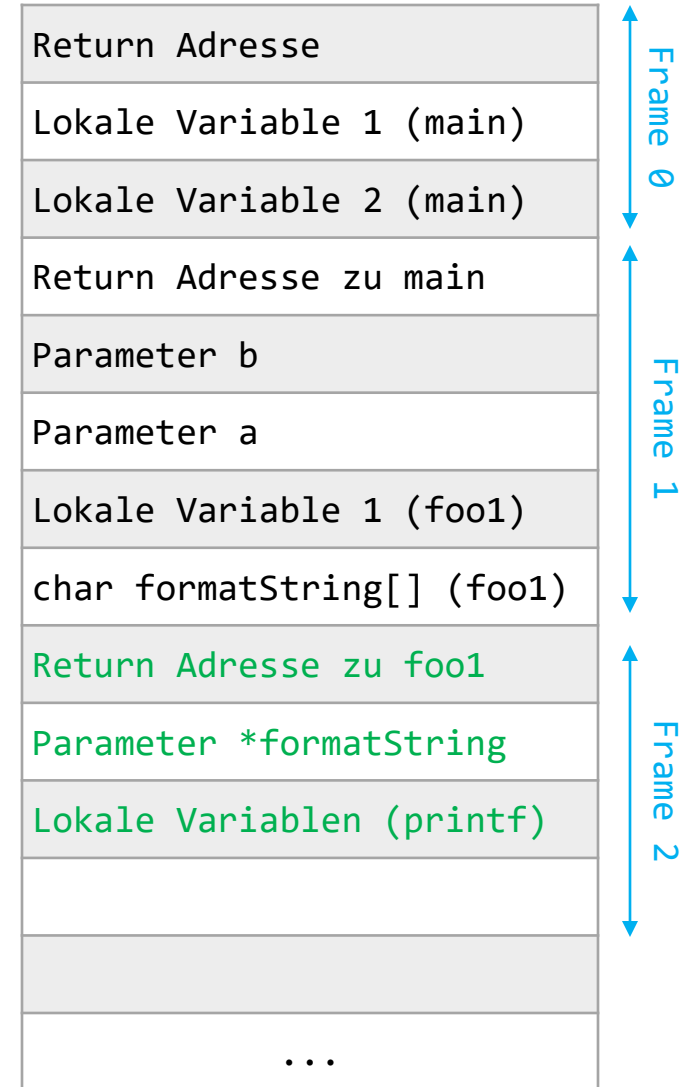
foo1(int a, int b)

printf(char  
\*formatString)

High address

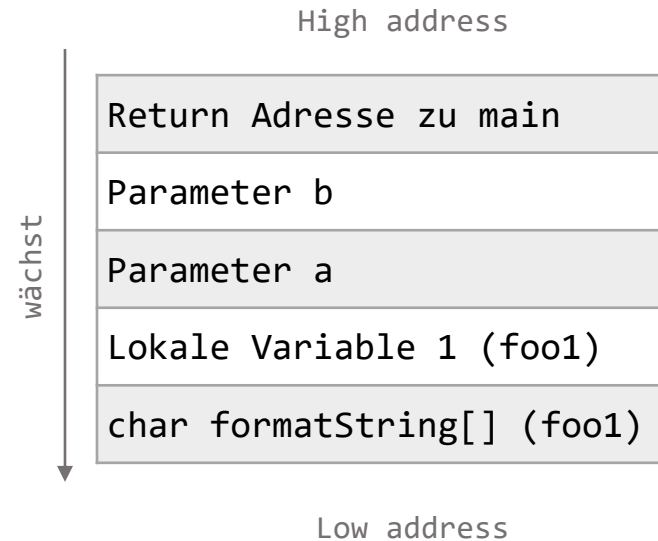
wächst  
↓

Low address



# Frame

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

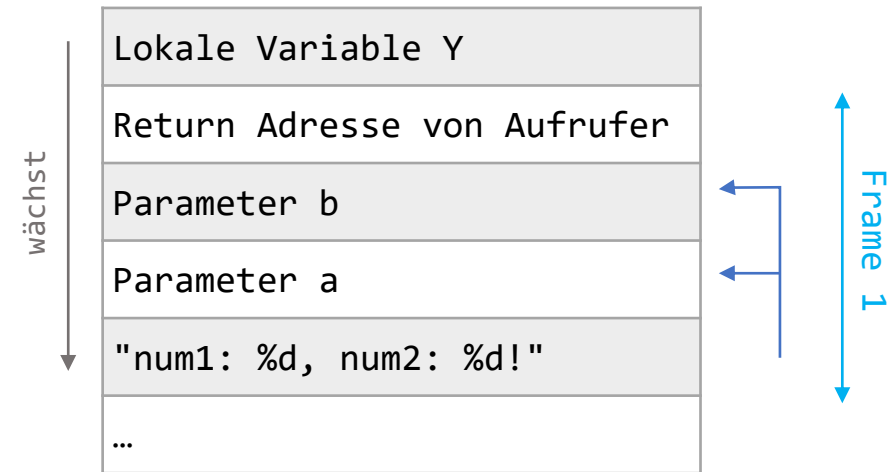


# Stack auslesen (1)

Hier: korrekte Anwendung von printf

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

High address

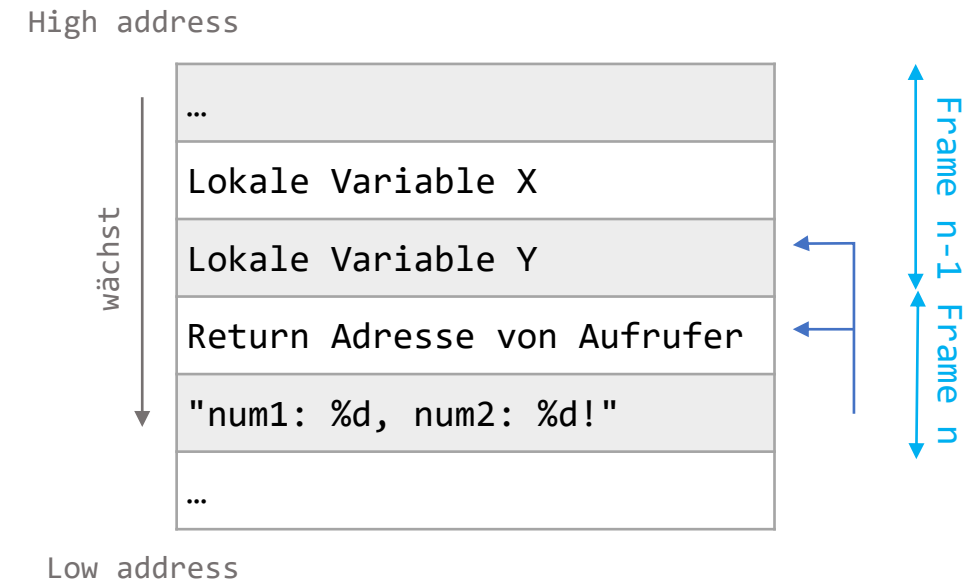


Low address

# 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



# 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, 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

# 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

# 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. "AAAABBBBB%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

# 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.

# 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)

# MITRE ATT&CK

# Weltweit zugängliche Wissensbasis über Taktiken und Techniken von Angreifern

Reconnaissance	Resource Development	Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Command and Control	Exfiltration	Impact
10 techniques	8 techniques	9 techniques	14 techniques	19 techniques	13 techniques	42 techniques	17 techniques	31 techniques	9 techniques	17 techniques	16 techniques	9 techniques	13 techniques
Active Scanning (3)	Acquire Access	Drive-by Compromise	Cloud Administration Command	Account Manipulation (5)	Abuse Elevation Control Mechanism (4)	Abuse Elevation Control Mechanism (4)	Adversary-in-the-Middle (3)	Account Discovery (4)	Exploitation of Remote Services	Adversary-in-the-Middle (3)	Application Layer Protocol (4)	Automated Exfiltration (1)	Account Access Removal
Gather Victim Host Information (4)	Acquire Infrastructure (8)	Exploit Public-Facing Application	Command and Scripting Interpreter (9)	BITS Jobs	Access Token Manipulation (5)	Access Token Manipulation (5)	Brute Force (4)	Application Window Discovery	Internal Spearphishing	Archive Collected Data (3)	Communication Through Removable Media	Data Transfer Size Limits	Data Destruction
Gather Victim Identity Information (3)	Compromise Accounts (3)	External Remote Services	Container Administration Command	Boot or Logon Autostart Execution (14)	Boot or Logon Autostart Execution (14)	BITS Jobs	Credentials from Password Stores (5)	Browser Information Discovery	Lateral Tool Transfer	Audio Capture		Exfiltration Over Alternative Protocol (3)	Data Encrypted for Impact
Gather Victim Network Information (6)	Compromise Infrastructure (7)			Boot or Logon Initialization Scripts (5)	Boot or Logon Initialization Scripts (14)	Build Image on Host		Cloud Infrastructure Discovery	Remote Service Session Hijacking (2)	Automated Collection	Data Encoding (2)		Data Manipulation (3)
Gather Victim Org Information (4)	Develop Capabilities (4)	Hardware Additions	Deploy Container		Boot or Logon Initialization Scripts (5)	Debugger Evasion	Exploitation for Credential Access	Cloud Service Dashboard		Browser Session Hijacking	Data Obfuscation (3)	Exfiltration Over C2 Channel	Defacement (2)
Phishing for Information (3)	Establish Accounts (3)	Phishing (3)	Exploitation for Client Execution	Browser Extensions	Create or Modify System Process (4)	Deobfuscate/Decode Files or Information	Forced Authentication	Cloud Service Discovery	Remote Services (7)	Clipboard Data	Dynamic Resolution (3)		Disk Wipe (2)
Search Closed Sources (2)	Obtain Capabilities (6)	Replication Through Removable Media	Inter-Process Communication (3)	Compromise Client Software Binary		Deploy Container	Forge Web Credentials (2)	Cloud Storage Object Discovery		Data from Cloud Storage	Encrypted Channel (2)	Exfiltration Over Other Network Medium (1)	Endpoint Denial of Service (4)
Search Open Technical Databases (5)	Stage Capabilities (6)	Supply Chain Compromise (3)	Native API	Create Account (3)	Domain Policy Modification (2)	Domain Policy Modification (2)	Input Capture (4)	Container and Resource Discovery	Replication Through Removable Media	Data from Configuration Repository (2)	Fallback Channels	Exfiltration Over Physical Medium (1)	Firmware Corruption
			Scheduled Task/Job (5)	Create or Modify System Process (4)	Event Triggered Execution (16)	Execution Guardrails (1)	Modify Authentication Process (8)	Debugger Evasion	Software Deployment Tools	Data from Information Repositories (3)	Ingress Tool Transfer	Exfiltration Over Web Service (3)	Inhibit System Recovery
Search Open Websites/Domains (3)		Trusted Relationship	Serverless Execution	Event Triggered Execution (16)	Exploitation for Privilege Escalation	Exploitation for Defense Evasion		Device Driver Discovery	Taint Shared Content	Data from Local System	Multi-Stage Channels		Network Denial of Service (2)
Search Victim-Owned Websites		Valid Accounts (4)	Shared Modules			File and Directory Permissions Modification (2)	Multi-Factor Authentication Interception	Domain Trust Discovery	Use Alternate Authentication Material (4)	Data from Network Shared Drive	Non-Application Layer Protocol	Scheduled Transfer	Resource Hijacking
			Software Deployment Tools	External Remote Services	Hijack Execution Flow (12)	Hide Artifacts (10)	Multi-Factor Authentication Request Generation	File and Directory Discovery			Non-Standard Port	Transfer Data to Cloud Account	Service Stop
			System Services (2)	Hijack Execution Flow (12)	Process Injection (12)	Hijack Execution Flow (12)	Network Sniffing	Group Policy Discovery					System Shutdown/Reboot
			User Execution (3)			Impair Defenses (10)		Network Service Discovery					
			Windows Management Instrumentation	Implant Internal Image	Scheduled Task/Job (5)	Indicator Removal (9)	OS Credential Dumping (8)	Network Share Discovery			Proxy (4)		
				Modify Authentication Process (8)	Valid Accounts (4)	Indirect Command Execution	Steal Application Access Tokens	Network Sniffing			Remote Access Software		
						Masking or Spoofing	Reversing Malware	Reversing Malware			Traffic		



# 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

# 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
```

# 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](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](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](https://en.wikipedia.org/wiki/Call_stack#Structure)
- [13] <https://www.youtube.com/watch?v=y5kcaqKYlql>

# Bildquellen (1)

- [1] [https://medium.com/@WriteupsTHM\\_HTB\\_CTF/cyber-kill-chain-tryhackme-7025c0662696](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/>