

Grundlagen der IT-Sicherheit

Übung 6. Softwaresicherheit



Übung zu Grundlagen der IT-Sicherheit, Wintersemester 2023/24

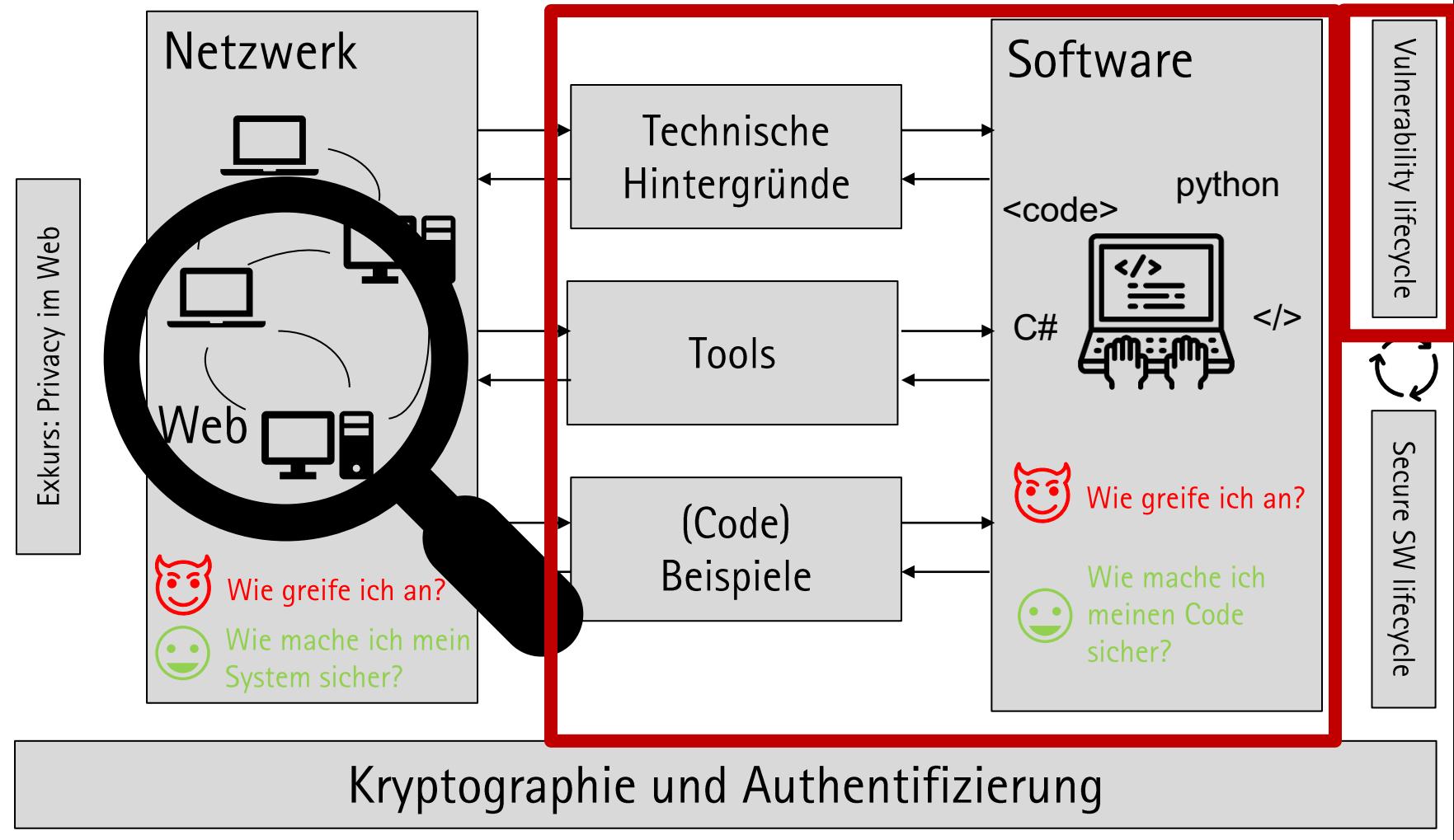
- Das Quiz 3 endet und sofort beginnt Quiz 4 ☺
- Raumwechsel für die Übung nächste Woche (18.12.2023)!
Raum F428 im Hauptgebäude
- Evaluation von Vorlesung und Übung laufen noch bis zum 17.12.
Nur durch Feedback können wir unsere Lehrveranstaltungen verbessern, also freuen wir uns über jegliche Rückmeldungen!

Unsere heutigen Themen...

- Grundlagen und Definitionen
 - Was ist eigentlich eine Verwundbarkeit
 - Lebenszyklus bei Verwundbarkeit
 - Stakeholder bei Verwundbarkeit
- Klassifikation von Verwundbarkeit
 - Benennung: CVE
 - Bewertung: CVSS
 - Organisation von Informationen
- Buffer-Overflows und Shellcode
- Exkurs: Meltdown und Spectre

Themen der Vorlesung

Grundlagen, allgemeine Prinzipien, Schutzziele, Faktor Mensch



GRUNDLAGEN UND DEFINITIONEN

Oder auch: Was ist eigentlich eine Verwundbarkeit?

Vulnerability:

Potenziell ausnutzbarer Fehler in einem System oder einer Software

- Grundlage für Angriff auf Vertraulichkeit, Integrität, Authentizität
- Wichtig: Nicht jeder Softwarefehler/-bug ist unbedingt für einen Angriff ausnutzbar

Woher kommen sie?

- Fehler im (System-) Entwurf
- Implementierungsfehler
- Fehlkonfigurationen des Systems
- Missbrauch oder fehlerhafte Anwendung durch die Nutzenden



https://commons.wikimedia.org/wiki/File:Heartbleed_bug_explained.svg

„Vulnerability“ im Englischen

- Schwachstelle
- Verwundbarkeit
- Sicherheitslücke

Meist synonym verwendet

- Kritischer Punkt: Öffentliche Bekanntgabe einer Sicherheitslücke („public disclosure“)
- Zero-Day Attack: Ausnutzung einer bisher unbekannten Sicherheitslücke

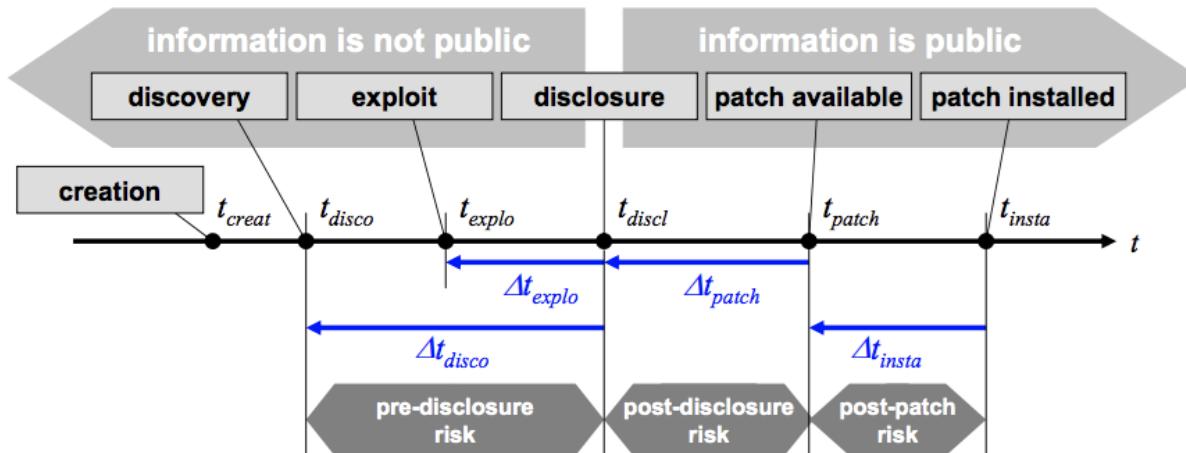
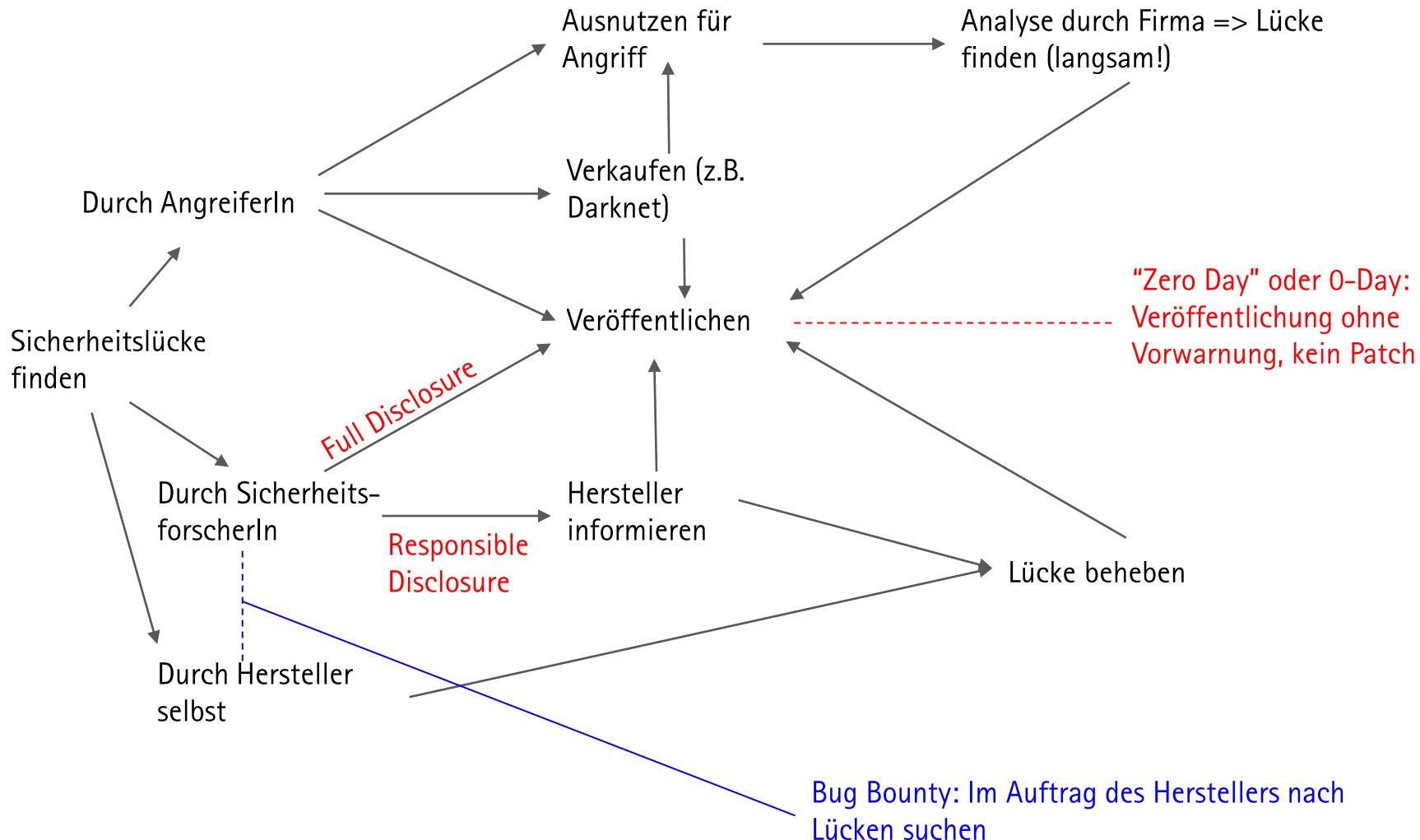


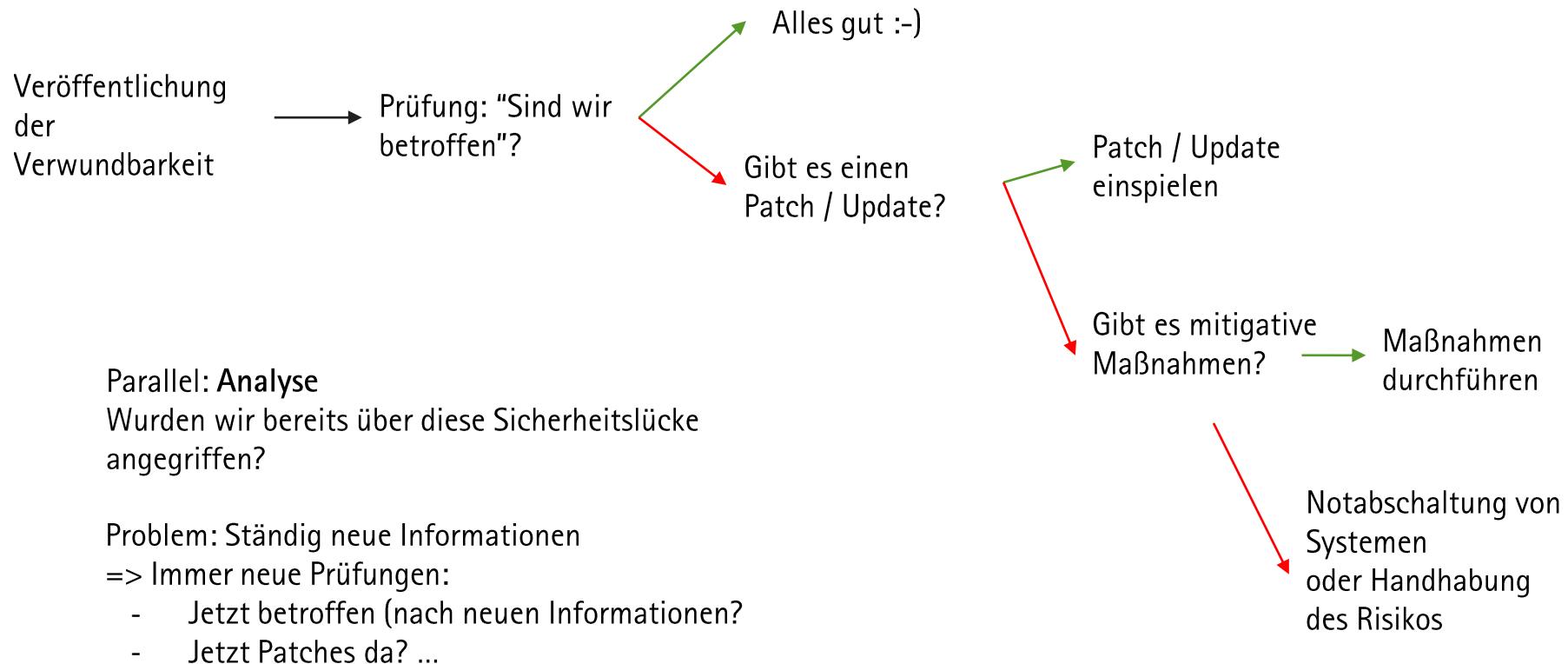
Figure 3: The lifecycle of a vulnerability defined by distinctive events. The exact sequence of events varies between vulnerabilities.

(Frei, WEIS 09 - <http://weis09.infosecon.net/files/103/paper103.pdf>)

Lebenszyklus – ein Blick auf die Praxis



Mit der Veröffentlichung einer Verwundbarkeit geht es für viele erst los ...

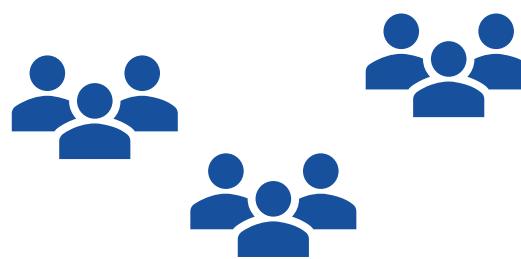


- Wird eine Verwundbarkeit gefunden, werden sehr viele Personengruppen aktiviert.
- Für einige davon gelten sie als "größter Alptraum". Für andere sind sie "wahr gewordener Wunschtraum".
- Wer ist alles betroffen und für wen gilt was?

Stakeholder bei Verwundbarkeiten



Herstellerfirma



Software-
Nutzende



Alptraum bei Herstellerfirmen:

- Management und Öffentlichkeitsarbeit
 - Bei Meldung einer Lücke muss quasi "sofort" eine Lösung bereitgestellt werden => Enormer öffentlicher Druck kombiniert mit Imageschaden
 - Meldepflichten / Druck durch EntdeckerIn der Verwundbarkeit

Alptraum bei Herstellerfirmen:

- Verantwortliche für die Applikationen: Schließung der Lücke organisieren
 - Ggf. Fremdfirmen erneut beauftragen, ggf. eigene Leute beauftragen.
 - Zeitkritisch - muss in bestehende Zeitpläne eingeschoben werden und "sofort" erledigt werden
 - Budgetplan: Kein Budget für abgeschlossene Projekte

Alpträum bei Herstellerfirmen:

- Entwicklerinnen und Entwickler: Verwundbarkeit beheben
 - Code wieder anfassen, der vor langer Zeit geschrieben wurde
 - Häufig Code bearbeiten, der von anderen geschrieben wurde ("geliehene" EntwicklerInnen, ...)

Alptraum bei Firmen, die die Software nutzen:

- IT-Sicherheitsbeauftragte: Bearbeitung der Lücke organisieren
Ständig neue Informationen erhalten und weitergeben, Betroffenheit prüfen, Patches prüfen und darüber Informieren, ...
 - Oft "Rechtfertigung" gegenüber Management nötig: Potenzielle Gefahr vs. Kosten aufrechnen
 - Zeitkritisch: alles "sofort" bearbeiten
 - Unklare Informationslage: Wenig klare Eingangskanäle, viele sich ständig verändernde Informationen, ...

Alptraum bei Firmen, die die Software nutzen:

- Verantwortliche für die Applikationen: Schließung der Lücke organisieren
 - Wenn Patch vorhanden: "Sofort" Patch einspielen (oft technische Probleme, unmögliche Zeitanforderungen, ...)
 - Wenn kein Patch vorhanden: Notfallpläne besprechen, ggf. Testen, ggf. Notabschaltung bewerten und durchführen lassen
 - Zeitkritisch – muss in bestehende Zeitpläne eingeschoben werden und "sofort" erledigt werden
 - Budgetplan: Kein Budget für abgeschlossene Projekte

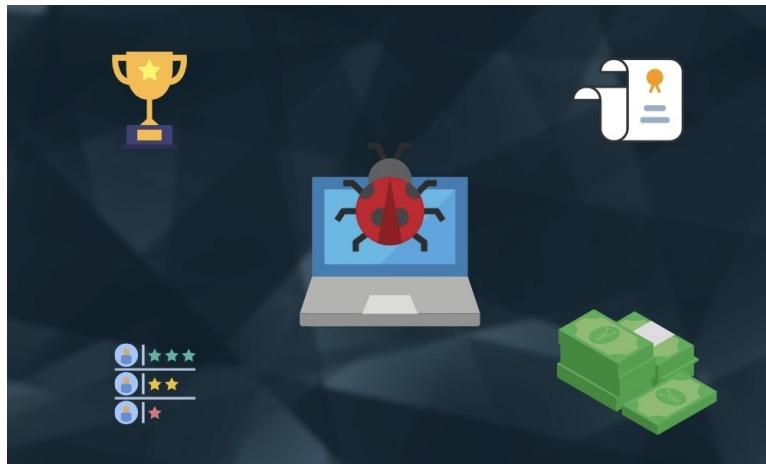
Wahr gewordener Traum:

- AngreiferInnen: Ausnutzen einer Lücke, die besteht und für die es (noch) keinen Schutz gibt.
Häufig sogar "Proof of Concept" und Anleitung veröffentlicht.

=> Wettrennen: Angriff oder Schutzmaßnahme zuerst?

Wahr gewordener Traum oder Alptraum:

- SicherheitsforscherIn:
 - Es KANN richtig Geld bringen, eine Sicherheitslücke zu finden.
 - Es KANN auch richtig Probleme geben, wenn man sie meldet.



Es KANN richtig Geld bringen, eine Sicherheitslücke zu finden.

Comparison of biggest bug bounty platforms:

Apple Security Bounty: rewards up to \$1,000,000 (a million dollars) for various security issues on iCloud and its smartphones.

Meta Bug Bounty (Whitehat): The reward money can go up to \$45,000. As per the bug's severity, the prize money can be a lot more (or a lot less).

Bug Hunters bounty program (Google): The rewards can go up to \$30,000 and more for special reports.

<https://geekflare.com/tech-companies-bug-bounty-programs/>

CDU-Connect-App

Danke für den Hinweis, Anzeige ist raus

Seite 3/3: Wie meldet man Sicherheitslücken, ohne sich strafbar zu machen?

<https://www.zeit.de/digital/datenschutz/2021-08/cdu-connect-app-it-sicherheit-lilith-wittmann-forscherin-klage/seite-3>

Es KANN auch richtig Probleme geben, wenn man sie meldet.

“Lilith Wittmann findet eine Sicherheitslücke in der Wahlkampf-App der CDU. Der Bundesgeschäftsführer bietet ihr einen Beratungsvertrag an. Die IT-Expertin lehnt ab - und wird angezeigt.”

<https://www.sueddeutsche.de/politik/cdu-connect-anzeige-wittmann-1.5373488>

“Nachdem Lilith Wittmann eine gravierende Sicherheitslücke in einer CDU-App entdeckt hatte, ermittelt nun das LKA gegen sie. Die CDU hatte sie angezeigt, doch die Anzeige jetzt nach öffentlichem Druck zurückgezogen.”

<https://netzpolitik.org/2021/cdu-connect-berliner-lka-ermittelt-gegen-it-expertin-die-sicherheitsluecken-in-partei-app-fand/>

Bug Bounty

Get Paid Up To
\$250,000 For
Hacking.



<https://thehackernews.com/2017/07/microsoft-bug-bounty-program.html>

Bug Bounty Programme ermöglichen "erlaubtes Hacken" mit fest definierter Bezahlung und festen Regeln

- Eigene Programme einzelner Firmen sowie Angebote, bei denen verschiedene Firmen beteiligt sind (z.B. Open Bug Bounty, Bugcrowd)
- Klare Angabe unter welchen Bedingungen gehackt werden darf und was angegriffen werden darf
- Feste Bezahlung für fest definierte Lücken (Art der Lücke, Kritikalität)
- Häufig "Spiel-Charakter": Punkte für Lücken, Ranglisten von Beteiligten, etc.

Positive Entwicklung: Sicherheitslücken bekommen eine gewisse Bekanntheit und werden zunehmend auch von der Öffentlichkeit und vom Management wahrgenommen.

Bsp. Bezahlung für gemeldete Lücken und "Aufschrei" in den Medien bei der Anzeige im Fall Wittmann.

Besonders kritische Sicherheitslücken zeigen besondere Kritikalität und fördern damit die allgemeine Wahrnehmung.

Bsp. Log4shell und seine Medienpräsenz

Beispiel Log4Shell

- Besonders kritische Sicherheitslücke, die über die Grenzen der IT-Sicherheit hinaus diskutiert wurde ("Das kam in der Tagesschau")
- Remote Code Execution vulnerability – AngreiferIn kann vollständige Kontrolle über betroffenes System erlangen
- Sicherheitslücke in Log4j Java logging library – Verbaut in tausenden von Applikationen. Ausführliche Prüfung notwendig, ob Systeme betroffen sind oder nicht.
- Erst kein Patch und keine Mitigationen bekannt. Dann Mitigationen. Feststellung: nicht ausreichend. Dann Patch. Feststellung: Ebenfalls Verwundbar (für neue Lücke). Neuer Patch. => Großes Chaos, was denn nun hilft und was nicht.
- Da Log4j in Applikationen eingebaut wurde, mussten häufig die Herausgeber dieser Applikationen selbst Updates bereitstellen.

A LIMITED VULNERABILITIES FOUND IN 2.15.0 AND 2.16.0

As of Tuesday, Dec 14, version 2.15.0 was found to still have a possible vulnerability in some apps. And, a few days later, a DOS vulnerability was found in 2.16.0 too.

We recommend updating to 2.16.0 which disables JNDI and completely removes \${lookup}.
(Updated: See below)

We recommend updating to 2.17.0 which includes the fixes introduced in 2.16.0 as well as a fix for a discovered denial of service (DOS) attack.

<https://www.lunasec.io/docs/blog/log4j-zero-day/>

Beispiel Log4Shell

Aussagekräftige "Meldekette":

Open Source Bibliothek - Frage im Forum zu "Fehler" mit tagelanger (langsamer) Diskussion.

Dann die Frage: "Ist das nicht auch eine Sicherheitslücke"?

Damit Präsenz im Bereich IT-Sicherheit - Präsenz in den Medien

Your next task is to figure out which applications in your org use log4j



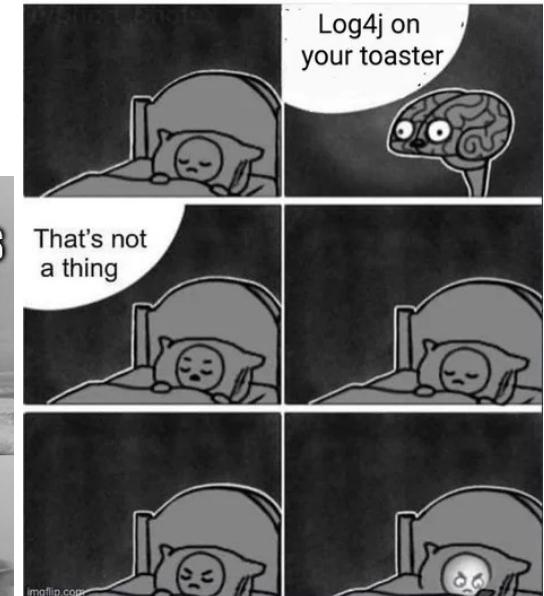
<https://humornama.com/memes/my-deepest-sympathies-to-java-developers/>



<https://www.linkedin.com/pulse/log4j-security-vulnerabilities-cve-2021-44228-mitigation-brown>



<https://securityboulevard.com/2021/12/log4j-the-meme-0/>



<https://9gag.com/tag/log4j/fresh>

KLASSIFIKATION VON VERWUNDBARKEITEN

Oder auch: Wie schlimm ist denn die Verwundbarkeit?

Benennung von Verwundbarkeiten

Wichtig: Klare Benennung, damit man sich über einzelne Schwachstellen austauschen kann.

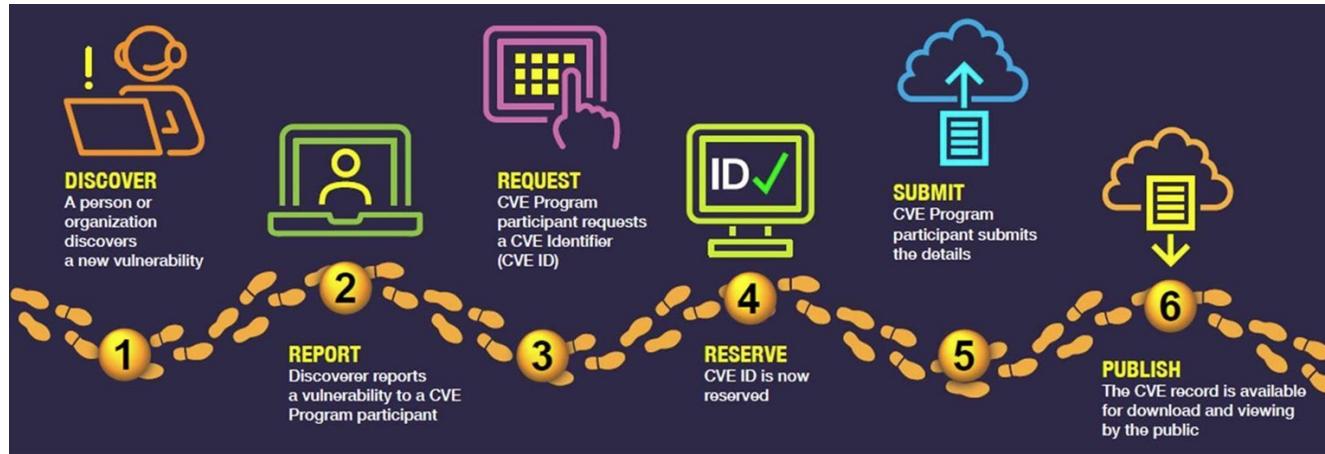
Doch nicht jede Schwachstelle hat einen Namen (z.B. Log4shell, Spectre, Meltdown, Printnightmare)

Alternativ Benennung von Firma, Produkt, Version und Art der Schwachstelle:
"Die Sicherheitslücke in Google Chrome in der Version 107.0.5304.121 die einen heap buffer overflow ermöglicht"

=> Referenziersystem CVE (Common Vulnerabilities and Exposures)

Einheitliche Benennung von Sicherheitslücken im IT-Bereich

Benennung von Schwachstellen



<https://www.cve.org/About/Process>

Aussehen:

CVE – Jahr – Nummer

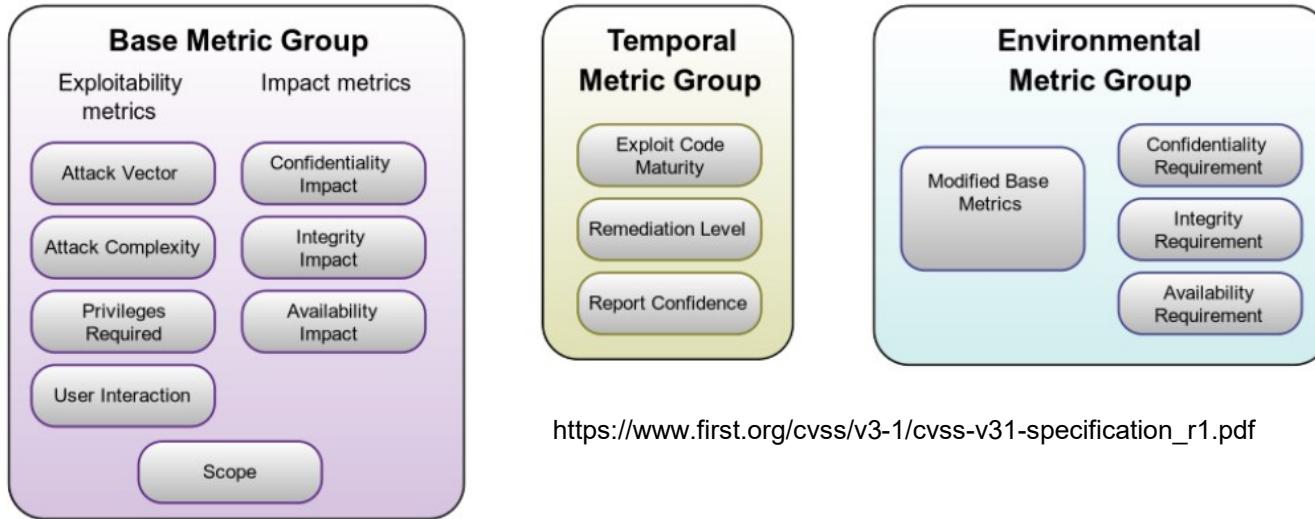
CVE-2021-44228

"Wie schlimm ist diese Schwachstelle?"

Allgemeines Bewertungssystem für Schwachstellen (Standard):
CVSS – Common Vulnerability Scoring System

Idee: Vergleichbarkeit und allgemeine Verständlichkeit durch feste Kriterien (Metrics)

Verschiedene Versionen: CVSS (2005), CVSSv2.0 (2007), CVSSv3.0 (2015)



https://www.first.org/cvss/v3-1/cvss-v31-specification_r1.pdf

Charakteristik der
Verwundbarkeit an sich -
konstant über Zeit und
zwischen verschiedenen
Umgebungen

Verändert sich über
die Zeit

Abhängig von der Umgebung
(z.B. firmeninterne
Gegebenheiten)

Online Berechnung in verschiedenen Online-Tools

Ein Beispiel:

<https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator>

Base Score Metrics

Exploitability Metrics

Attack Vector (AV)*

Network (AV:N) Adjacent Network (AV:A) Local (AV:L) Physical (AV:P)

Attack Complexity (AC)*

Low (AC:L) High (AC:H)

Privileges Required (PR)*

None (PR:N) Low (PR:L) High (PR:H)

User Interaction (UI)*

None (UI:N) **Required (UI:R)**

Scope (S)*

Unchanged (S:U) Changed (S:C)

Impact Metrics

Confidentiality Impact (C)*

None (C:N) Low (C:L) High (C:H)

Integrity Impact (I)*

None (I:N) Low (I:L) High (I:H)

Availability Impact (A)*

None (A:N) Low (A:L) High (A:H)

Temporal Score Metrics

Exploitability (E)

Not Defined (E:X)	Unproven that exploit exists (E:U)	Proof of concept code (E:P)	Functional exploit exists (E:F)	High (E:H)
-------------------	------------------------------------	------------------------------------	---------------------------------	------------

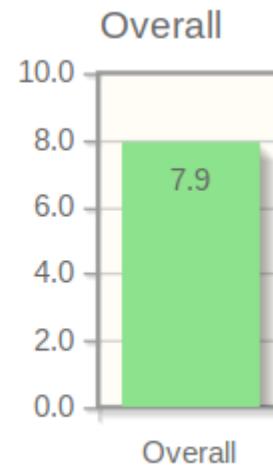
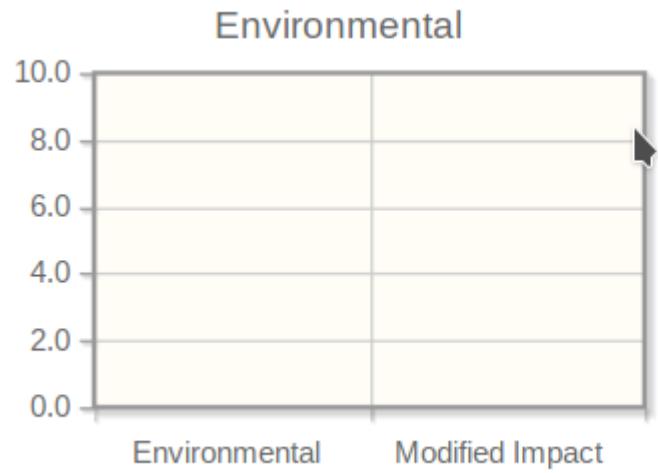
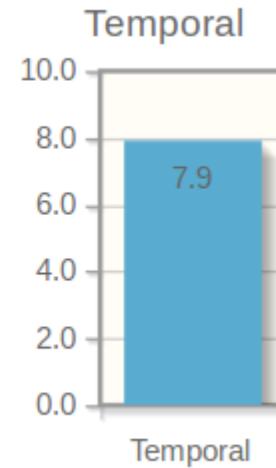
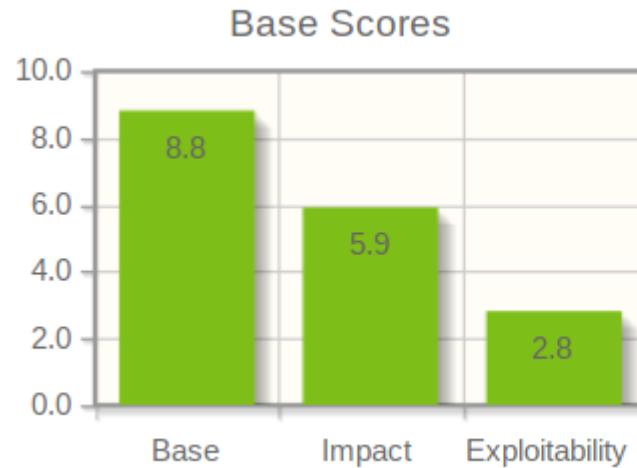
Remediation Level (RL)

Not Defined (RL:X)	Official fix (RL:O)	Temporary fix (RL:T)	Workaround (RL:W)	Unavailable (RL:U)
--------------------	----------------------------	----------------------	-------------------	--------------------

Report Confidence (RC)

Not Defined (RC:X)	Unknown (RC:U)	Reasonable (RC:R)	Confirmed (RC:C)
--------------------	----------------	-------------------	-------------------------

Bewertung von Verwundbarkeiten



CVSS Base Score: 8.8

Impact Subscore: 5.9

Exploitability Subscore: 2.8

CVSS Temporal Score: 7.9

CVSS Environmental Score: NA

Modified Impact Subscore: NA

Overall CVSS Score: 7.9

Bewertung von Verwundbarkeiten

„Wie schlimm ist diese Schwachstelle?“ => CVSS

Weitere Arten der Bewertung sind möglich.
Beispiele vom BSI:

				SCHADENSPOTENTIAL				
				Verlust	Kontext			
					Benutzer	Dienst	System	Netzwerk
DRINGLICHKEIT / EINTRITTPOTENZIAL				Übernahme der Kontrolle	hoch	hoch	sehr hoch	sehr hoch
Status der Schwachstelle		Verbreitungsmethode		Übernahme von Berechtigungen	mittel	mittel	hoch	hoch
theoretisch		manuell	automatisch	replizierend	Integrität	gering	mittel	hoch
ausnutzbar		sehr gering	gering	mittel	Vertraulichkeit	sehr gering	gering	mittel
aktiv		gering	mittel	hoch	Verfügbarkeit	sehr gering	gering	mittel
Exploit veröffentlicht		mittel	hoch	sehr hoch	Umgehung von Sicherheitsmaßnahmen	sehr gering	gering	mittel

<https://www.bsi.bund.de/EN/Service-Navi/Abonnements/Newsletter/Buerger-CERT-Abos/Buerger-CERT-Sicherheitshinweise/Risikostufen/risikostufen.html>

Organisation von Informationen

Und in der Praxis...?

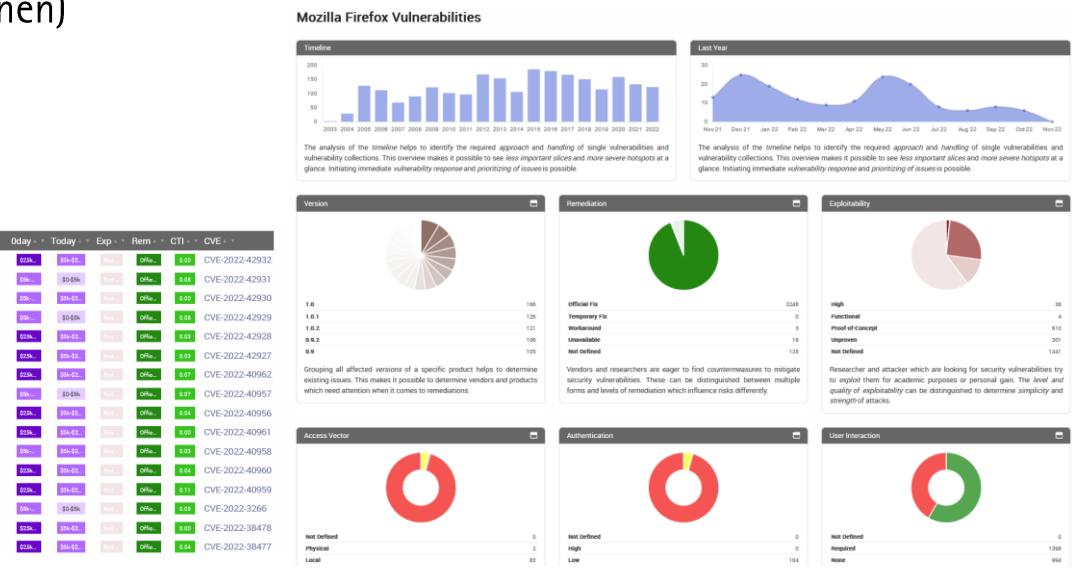
Riesige Datenbanken mit Informationen über die jeweiligen Schwachstellen

- CVE
- Welche Hersteller | Produkt | Version betroffen?
- CVSS (meist unterschiedliche Versionen)
- Patch / Mitigation / Fix vorhanden?

Ein Beispiel: <https://vuldb.com>

Published	Base	Temp	Vulnerability
10/18/2022	4.1	4.0	Mozilla Firefox memory corruption
10/18/2022	4.2	4.0	Mozilla Firefox Form Manager cleartext storage in a file or on disk
10/18/2022	0.0	4.0	Mozilla Firefox ThirdPartyUtil race condition
10/18/2022	4.0	4.0	Mozilla Firefox window print denial of service
10/18/2022	0.1	4.0	Mozilla Firefox Garbage Collector memory corruption
10/18/2022	4.5	4.0	Mozilla Firefox unknown vulnerability
09/20/2022	4.0	4.0	Mozilla Firefox memory corruption
09/20/2022	0.1	4.0	Mozilla Firefox denial of service
09/20/2022	4.0	4.0	Mozilla Firefox Content-Security-Policy injection
09/20/2022	4.1	4.0	Mozilla Firefox Graphics Driver stack-based overflow
09/20/2022	0.1	4.0	Mozilla Firefox Cookie session fixation
09/20/2022	4.1	4.0	Mozilla Firefox URL Parser use after free
09/20/2022	4.1	4.0	Mozilla Firefox FeaturePolicy access control
09/20/2022	4.0	4.0	Mozilla Firefox H264 Decoder out-of-bounds
08/23/2022	4.0	4.0	Mozilla Firefox memory corruption
08/23/2022	4.0	4.0	Mozilla Firefox memory corruption

Liste von Verwundbarkeiten für dieses Produkt mit Informationen



Statistik je nach Produkt

Organisation von Informationen

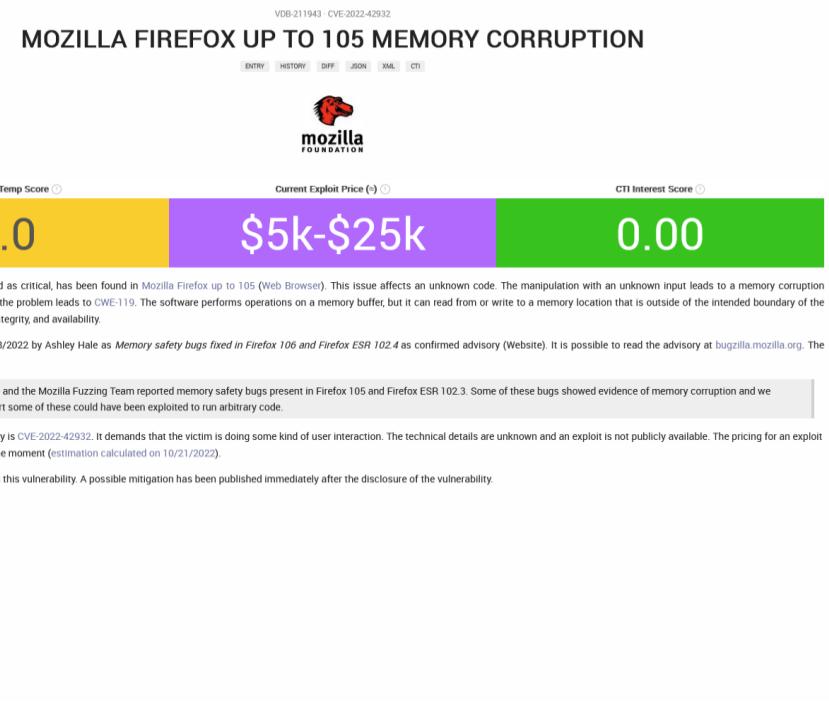
Und in der Praxis...?

Riesige Datenbanken mit Informationen über die jeweiligen Schwachstellen

- CVE
- Welche Hersteller | Produkt | Version betroffen?
- CVSS (meist unterschiedliche Versionen)
- Patch / Mitigation / Fix vorhanden?

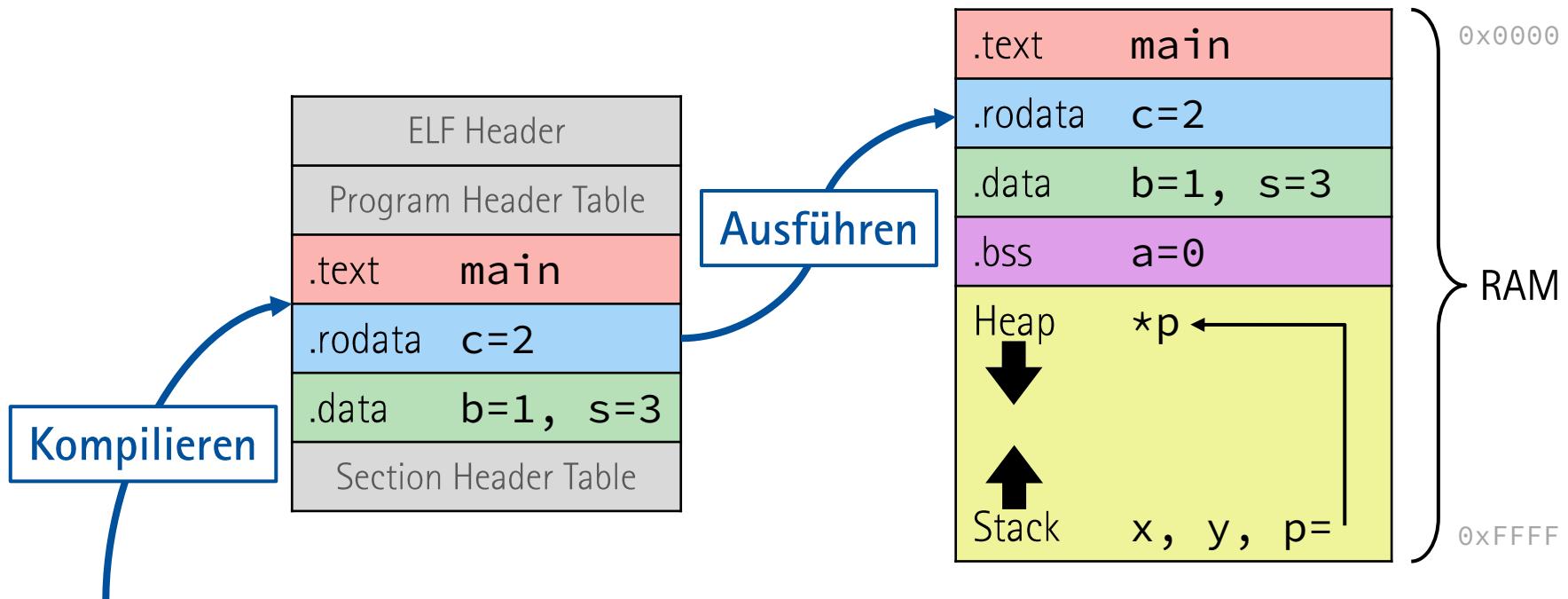
Ein Beispiel: <https://vuldb.com>

Information zu jeder einzelnen Schwachstelle



BUFFER-OVERFLOWS UND SHELLCODE

Prozesslayout: Stack und Heap



```
int a;                                // a: global, uninitialized
int b = 1;                             // b: global, initialized
const int c = 2;                         // c: global, const

void main() {
    static int s = 3;                   // s: local, static, initialized
    int x, y;                          // x: local, auto; y: local, auto
    char *p = malloc(100); // p: local, auto; *p: heap (100 byte)
}
```

Prozesslayout: Stack im Detail



```
int main() {
    random(time(NULL));
    int number = random() % 100; // Choose random number

    char guess[3]; // Space for user guess

    printf("I'm thinking of a number between 0 and 99.\n");
    printf("Can you guess it? (Hint: It is %d)\n", number);

    while (true) { // Game loop
        scanf("%s", guess);

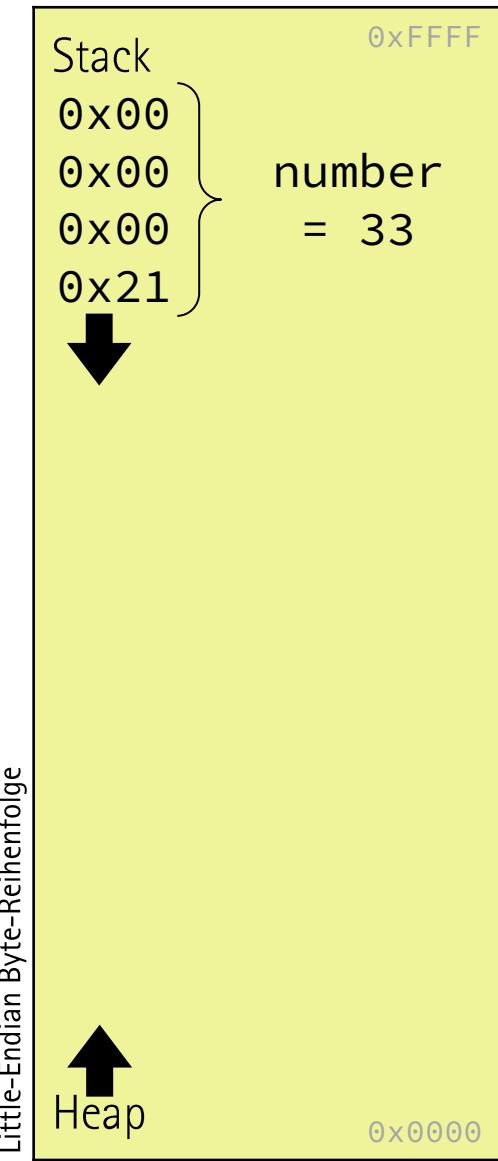
        if (number == atoi(guess))
            break;

        printf("My number is %s.",
               (number > atoi(guess)) ? "bigger" : "smaller"));
        printf("Try again:\n");
    }

    printf("Success! You have correctly guessed my number.");
    printf("It was %d.\n", number);
    return 0;
}
```

Achtung:
Speicher ist im Vergleich zur
vorherigen Folie „umgedreht“

Prozesslayout: Stack im Detail



Angenommen:

number = 33 // 0x21
guess = "AB"

```
int main() {
    random(time(NULL));
    int number = random() % 100; // Choose random number

    char guess[3]; // Space for user guess

    printf("I'm thinking of a number between 0 and 99.\n");
    printf("Can you guess it? (Hint: It is %d)\n", number);

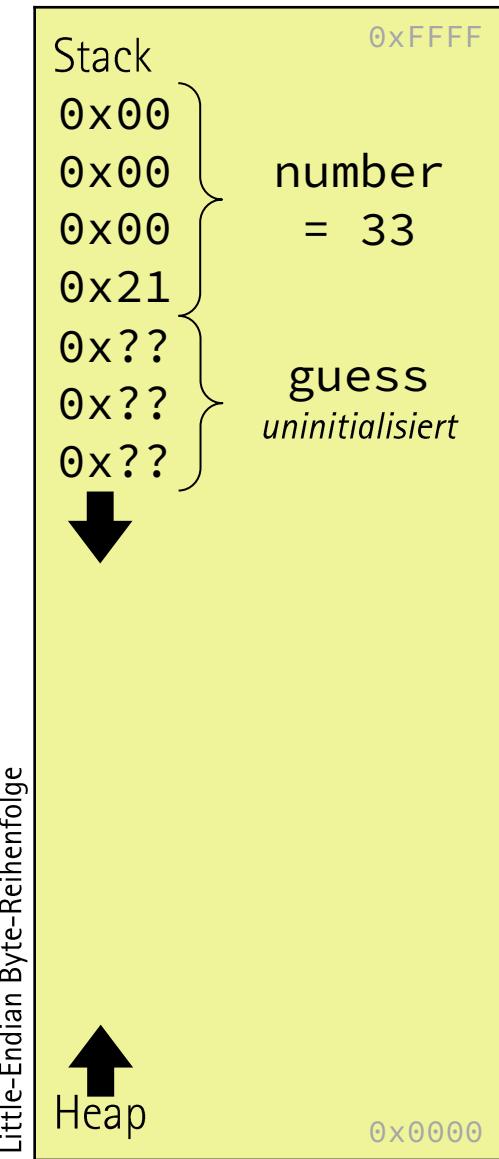
    while (true) { // Game loop
        scanf("%s", guess);

        if (number == atoi(guess))
            break;

        printf("My number is %s.",
               (number > atoi(guess)) ? "bigger" : "smaller"));
        printf("Try again:\n");
    }

    printf("Success! You have correctly guessed my number.");
    printf("It was %d.\n", number);
    return 0;
}
```

Prozesslayout: Stack im Detail



Angenommen:

number = 33 // 0x21
guess = "AB"

```
int main() {
    random(time(NULL));
    int number = random() % 100; // Choose random number

    char guess[3]; // Space for user guess
    printf("I'm thinking of a number between 0 and 99.\n");
    printf("Can you guess it? (Hint: It is %d)\n", number);

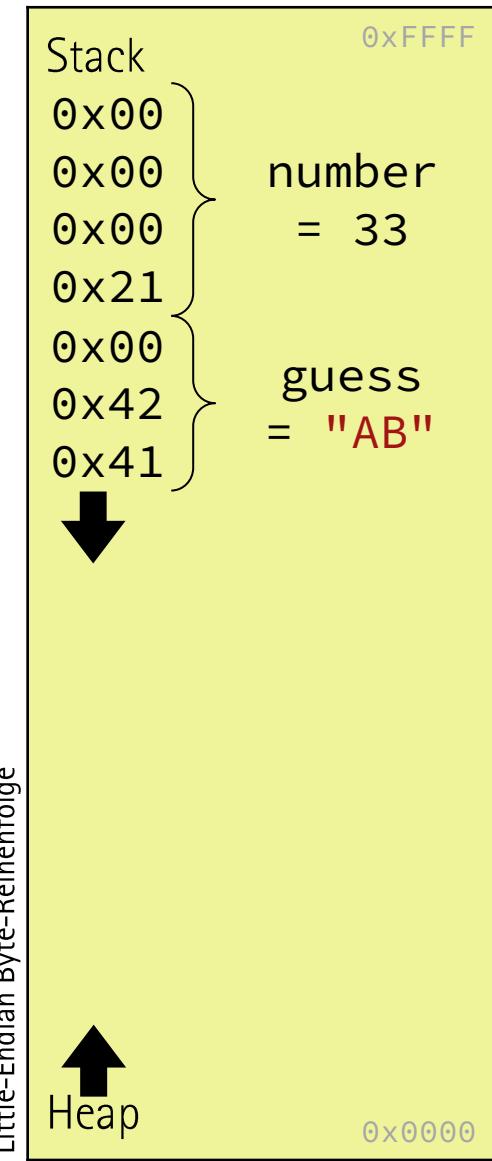
    while (true) { // Game loop
        scanf("%s", guess);

        if (number == atoi(guess))
            break;

        printf("My number is %s.",
               (number > atoi(guess)) ? "bigger" : "smaller"));
        printf("Try again:\n");
    }

    printf("Success! You have correctly guessed my number.");
    printf("It was %d.\n", number);
    return 0;
}
```

Prozesslayout: Stack im Detail



Angenommen:

`number = 33 // 0x21`
`guess = "AB"`

```
int main() {
    random(time(NULL));
    int number = random() % 100; // Choose random number

    char guess[3]; // Space for user guess

    printf("I'm thinking of a number between 0 and 99.\n");
    printf("Can you guess it? (Hint: It is %d)\n", number);

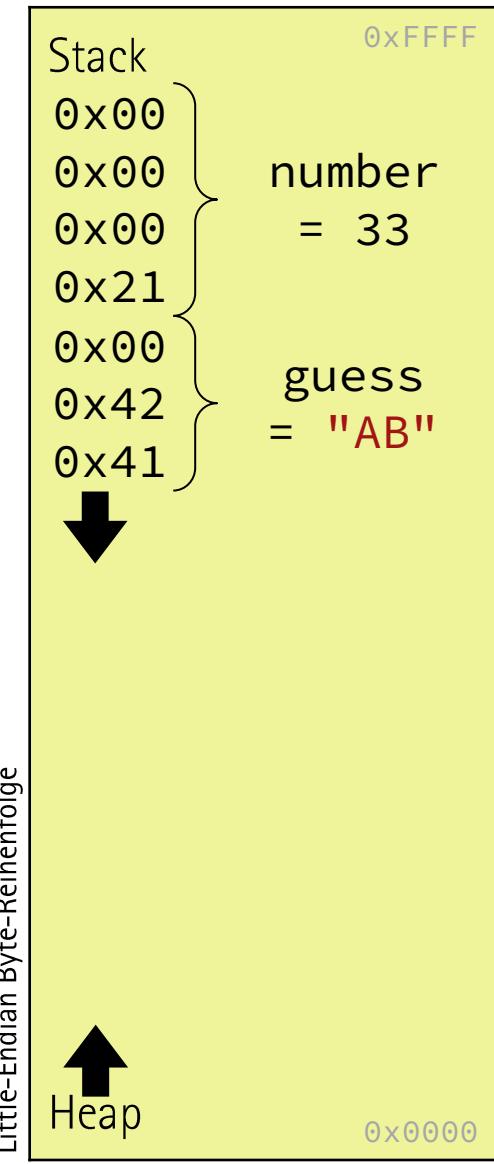
    while (true) { // Game loop
        scanf("%s", guess);

        if (number == atoi(guess))
            break;

        printf("My number is %s.",
               (number > atoi(guess)) ? "bigger" : "smaller"));
        printf("Try again:\n");
    }

    printf("Success! You have correctly guessed my number.");
    printf("It was %d.\n", number);
    return 0;
}
```

Prozesslayout: Stack im Detail



```
int main() {
    random(time(NULL));
    int number = random() % 100; // Choose random number

    char guess[3]; // Space for user guess

    printf("I'm thinking of a number between 0 and 99.\n");
    printf("Can you guess it? (Hint: It is %d)\n", number);

    while (true) { // Game loop
        scanf("%s", guess);

        if (number == atoi(guess))
            break;

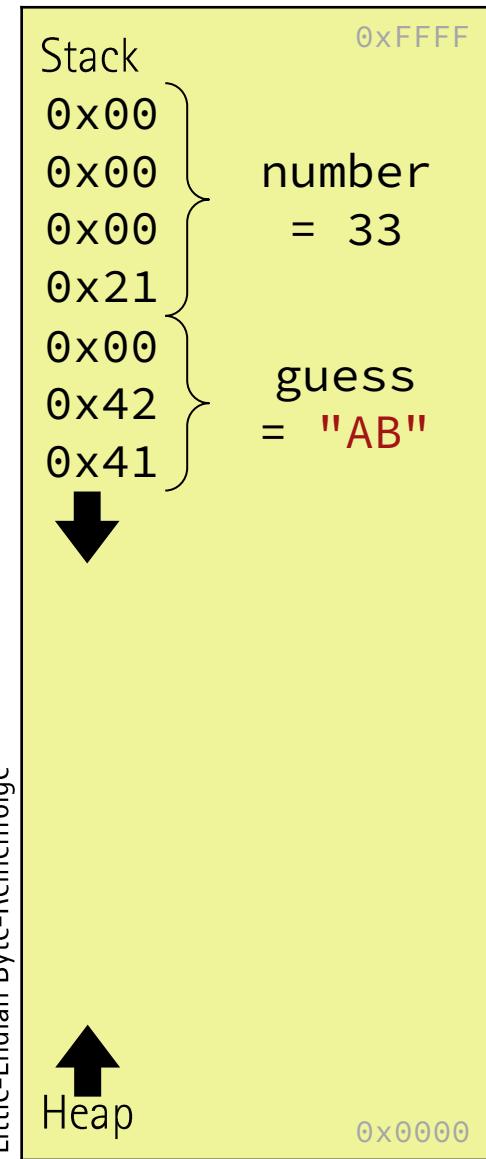
        printf("My number is %s.",
               (number > atoi(guess)) ? "bigger" : "smaller"));
        printf("Try again:\n");
    }

    printf("Success! You have correctly guessed my number.");
    printf("It was %d.\n", number);
    return 0;
}
```

Gibt es eine Eingabe mit der man immer gewinnt?

Hinweis:
`atoi()` hört beim ersten nicht-Ziffer-Zeichen auf.

Prozesslayout: Stack-Overflow



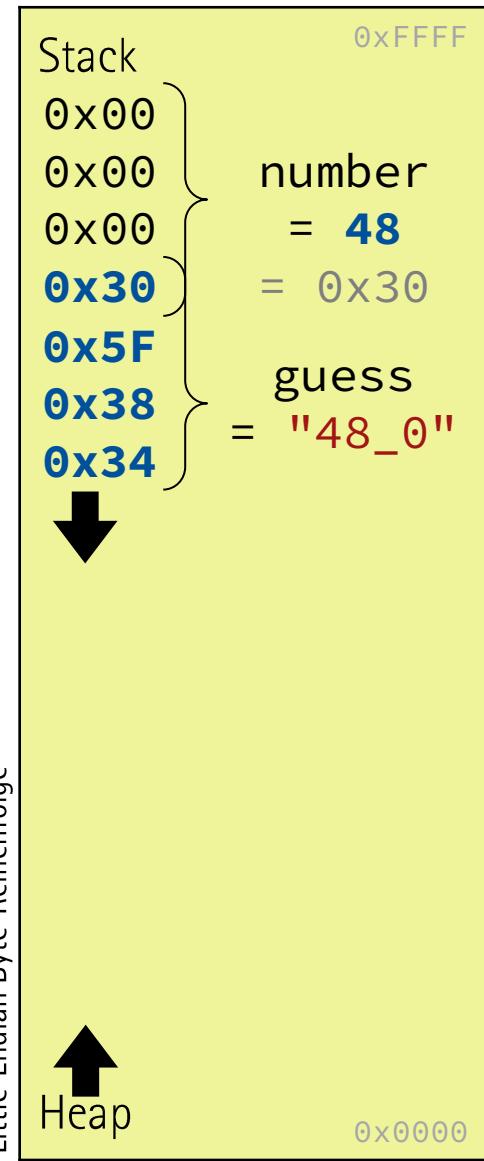
int main() {
 random(time(NULL));
 int number = random() % 100; // Choose random number

 char guess[3]; // Space for user guess

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1x	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2x	SP	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	-
6x	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

Gibt es eine Eingabe mit der man immer gewinnt?

Prozesslayout: Stack-Overflow



int main() {
 srand(time(NULL));
 int number = random() % 100; // Choose random number

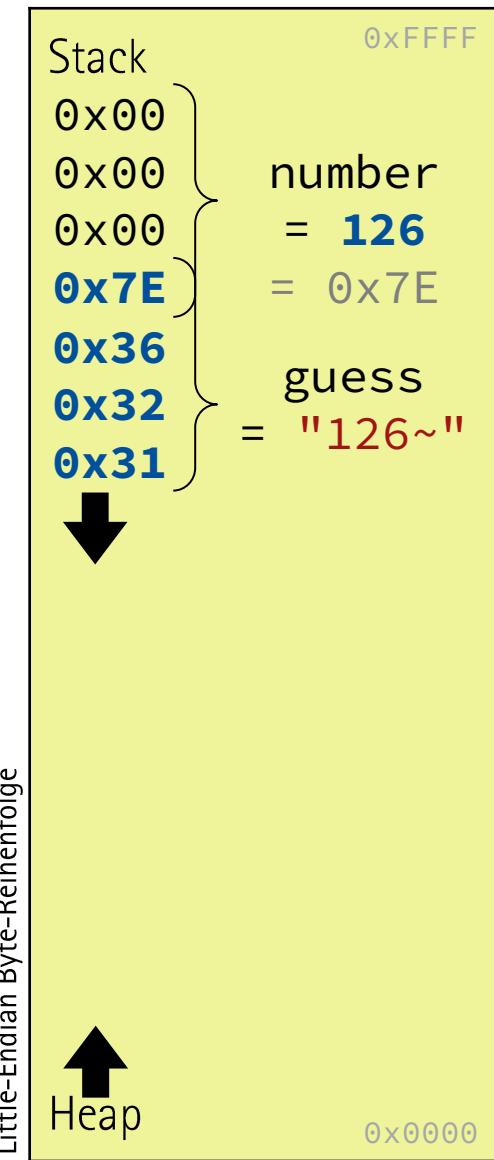
 char guess[3]; // Space for user guess

Gibt es eine Eingabe mit der man immer gewinnt?



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1x	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2x	SP	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6x	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

Prozesslayout: Stack-Overflow



```
int main() {  
    random(time(NULL));  
    int number = random() % 100; // Choose random number  
  
    char guess[3]; // Space for user guess
```

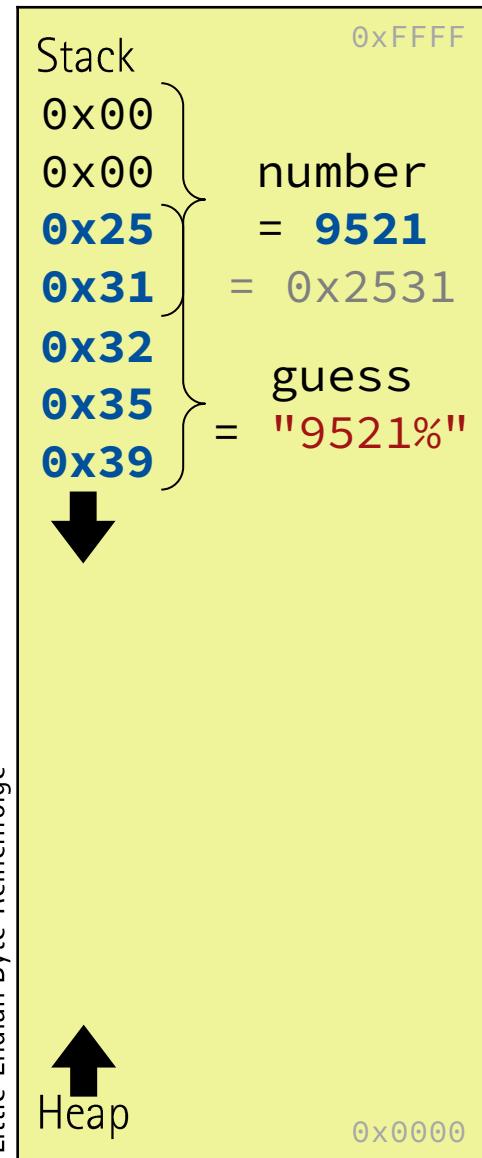
Gibt es eine Eingabe mit der man immer gewinnt?



Mögliche Eingaben:

33	!	= 0x21 = 33
34	"	= 0x22 = 34
35	#	= 0x23 = 35
...
97	a	= 0x61 = 97
98	b	= 0x62 = 98
99	c	= 0x63 = 99
100	d	= 0x64 = 100
101	e	= 0x65 = 101
...
126	~	= 0x7E = 126

Prozesslayout: Stack-Overflow



```
int main() {
    srand(time(NULL));
    int number = random() % 100; // Choose random number

    char guess[3]; // Space for user guess
```

Gibt es eine Eingabe mit der man immer gewinnt?



Mögliche Eingaben (Fortsetzung):

9520%	{	'9'	= 0x39
9521%		'5'	= 0x35
9522%		'2'	= 0x32
9523%		'1'	= 0x31
		'%'	= 0x25 } 0x2531

= 9521

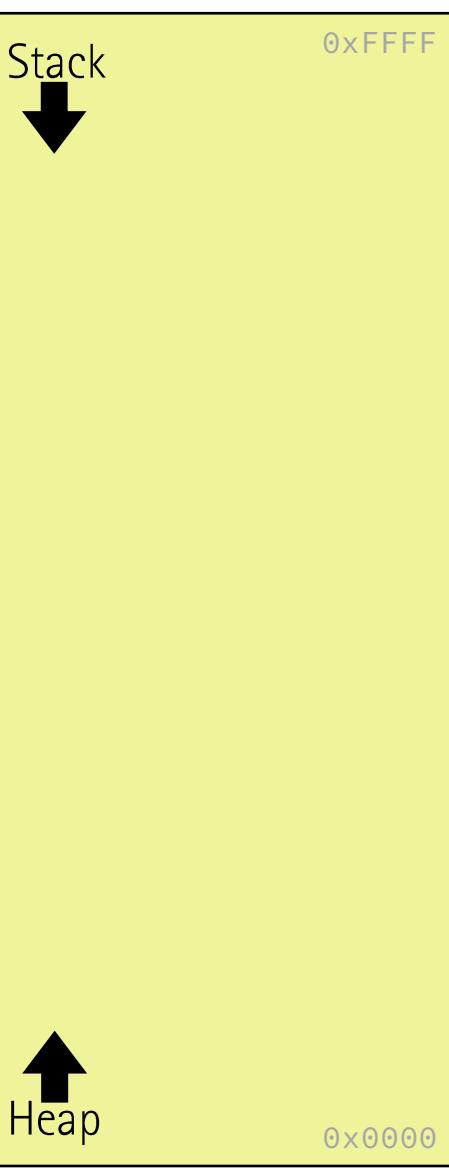
...

9528%
9529%

Hinweis: Der Quelltext für dieses Beispiel ist zusammen mit einem Makefile auf Stud.IP verfügbar

Prozesslayout: Stack im Detail

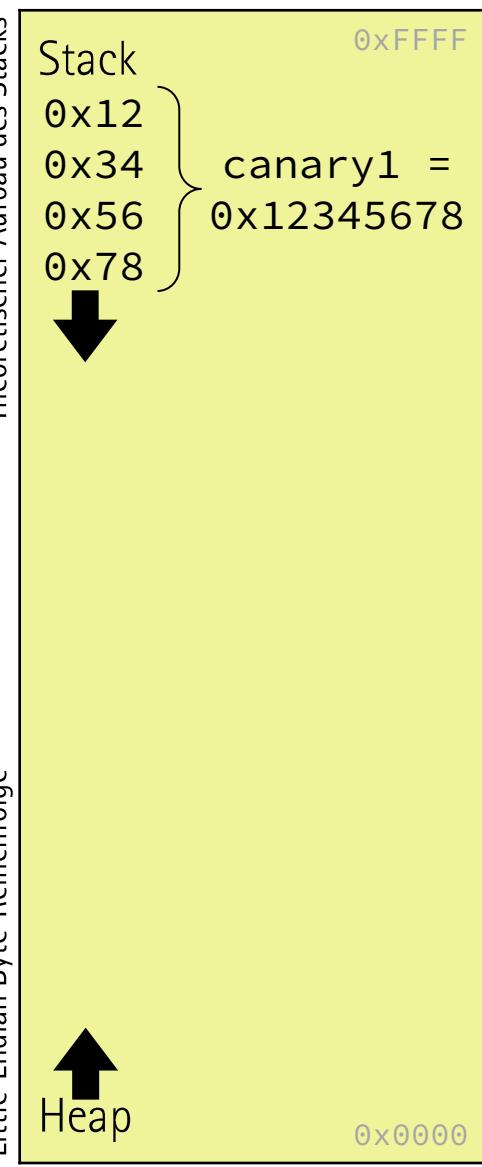
Theoretischer Aufbau des Stacks



```
bool check_password() {  
    volatile unsigned int canary1 = 0x12345678;  
    volatile char root_password[] = "secret";  
    volatile unsigned int canary2 = 0xDEADBEEF;  
    volatile char user_password[16] = "0123456789ABCDE";  
    volatile unsigned int canary3 = 0x87654321;  
  
    printf("Password (Hint it is '%s'): ", root_password);  
  
    scanf("%s", user_password);  
  
    if (canary1 != 0x12345678 || canary2 != 0xDEADBEEF  
        || canary3 != 0x87654321) {  
        printf("Stack manipulation detected!\n");  
        return false;  
    }  
  
    return strcmp(root_password, user_password) == 0;  
}
```

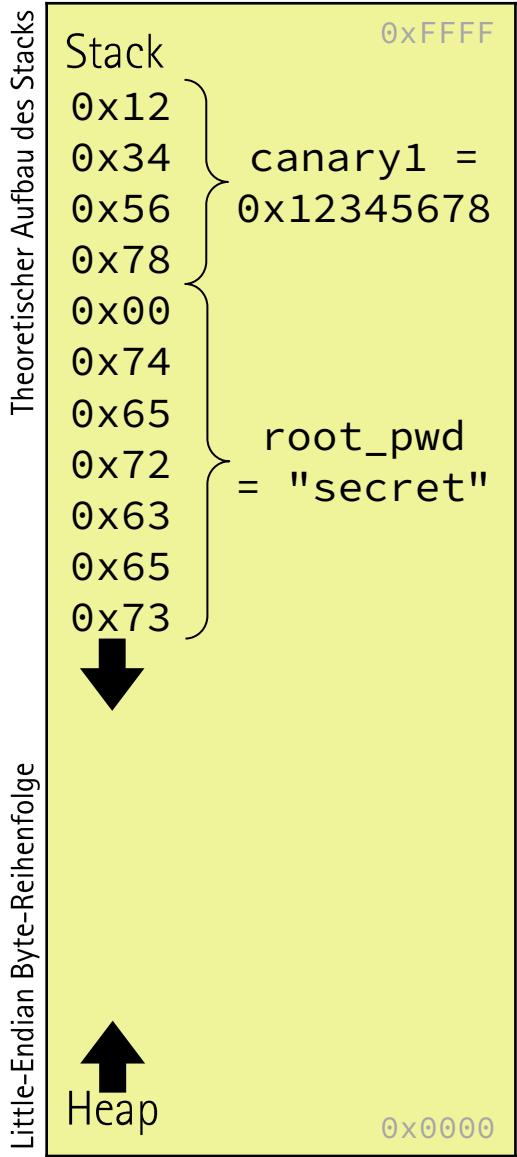
Prozesslayout: Stack im Detail

Theoretischer Aufbau des Stacks



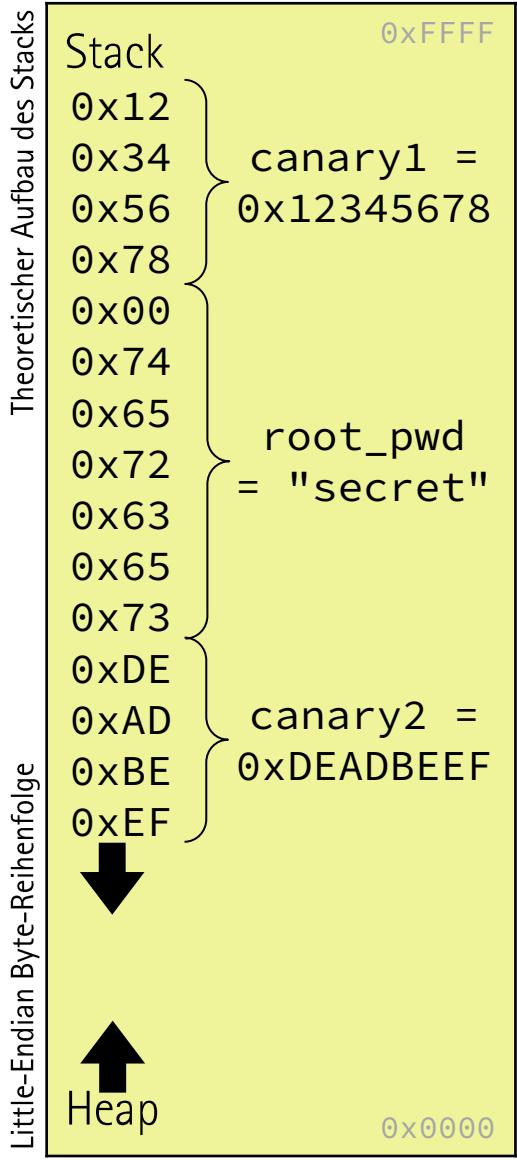
```
bool check_password() {  
    volatile unsigned int canary1 = 0x12345678;  
    volatile char root_password[] = "secret";  
    volatile unsigned int canary2 = 0xDEADBEEF;  
    volatile char user_password[16] = "0123456789ABCDE";  
    volatile unsigned int canary3 = 0x87654321;  
  
    printf("Password (Hint it is '%s'): ", root_password);  
  
    scanf("%s", user_password);  
  
    if (canary1 != 0x12345678 || canary2 != 0xDEADBEEF  
        || canary3 != 0x87654321) {  
        printf("Stack manipulation detected!\n");  
        return false;  
    }  
  
    return strcmp(root_password, user_password) == 0;  
}
```

Prozesslayout: Stack im Detail



```
bool check_password() {  
    volatile unsigned int canary1 = 0x12345678;  
    volatile char root_password[] = "secret";  
    volatile unsigned int canary2 = 0xDEADBEEF;  
    volatile char user_password[16] = "0123456789ABCDE";  
    volatile unsigned int canary3 = 0x87654321;  
  
    printf("Password (Hint it is '%s'): ", root_password);  
  
    scanf("%s", user_password);  
  
    if (canary1 != 0x12345678 || canary2 != 0xDEADBEEF  
        || canary3 != 0x87654321) {  
        printf("Stack manipulation detected!\n");  
        return false;  
    }  
  
    return strcmp(root_password, user_password) == 0;  
}
```

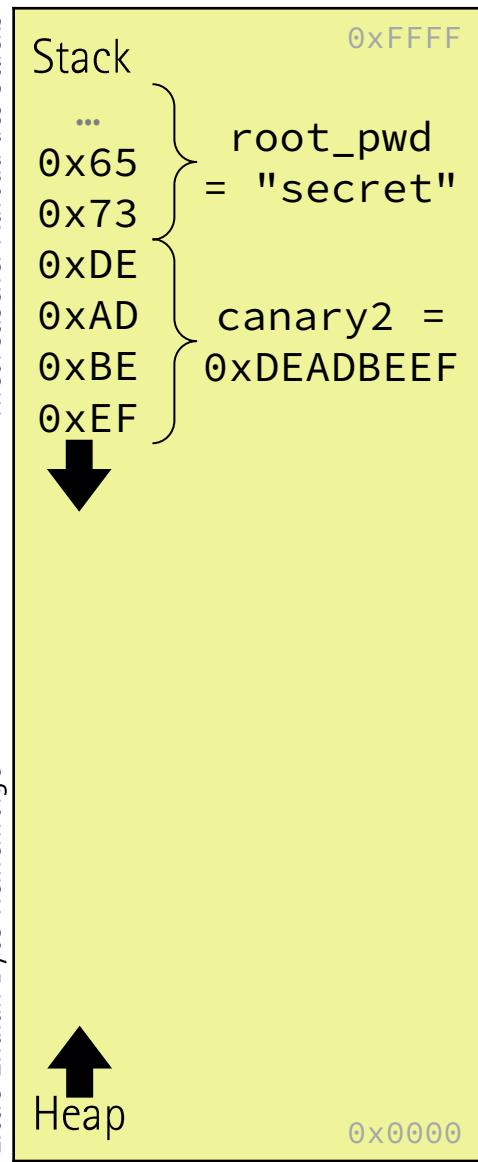
Prozesslayout: Stack im Detail



```
bool check_password() {  
    volatile unsigned int canary1 = 0x12345678;  
    volatile char root_password[] = "secret";  
    volatile unsigned int canary2 = 0xDEADBEEF;  
    volatile char user_password[16] = "0123456789ABCDE";  
    volatile unsigned int canary3 = 0x87654321;  
  
    printf("Password (Hint it is '%s'): ", root_password);  
  
    scanf("%s", user_password);  
  
    if (canary1 != 0x12345678 || canary2 != 0xDEADBEEF  
        || canary3 != 0x87654321) {  
        printf("Stack manipulation detected!\n");  
        return false;  
    }  
  
    return strcmp(root_password, user_password) == 0;  
}
```

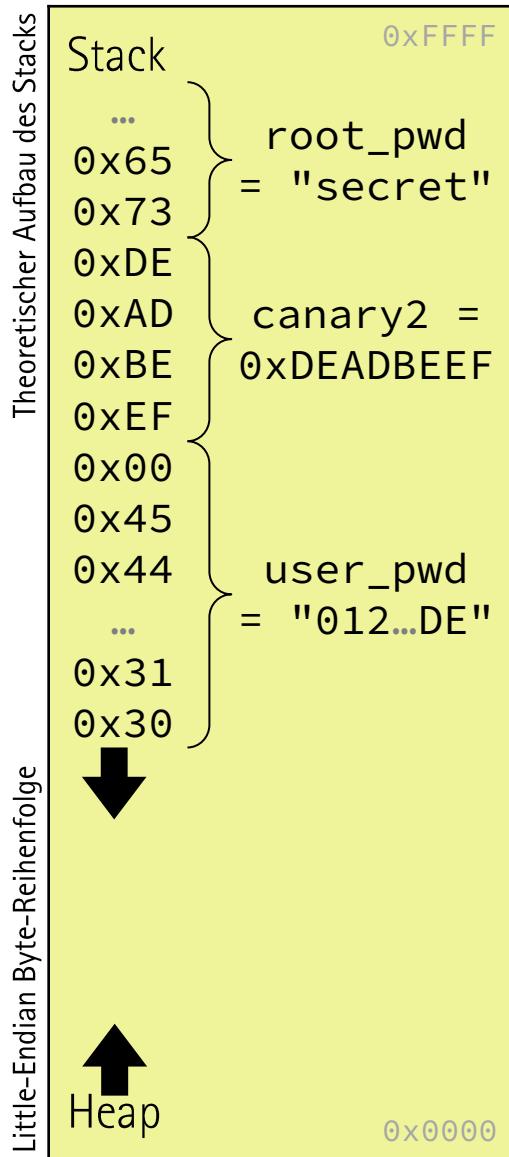
Prozesslayout: Stack im Detail

Theoretischer Aufbau des Stacks



```
bool check_password() {  
    volatile unsigned int canary1 = 0x12345678;  
    volatile char root_password[] = "secret";  
    volatile unsigned int canary2 = 0xDEADBEEF;  
    volatile char user_password[16] = "0123456789ABCDE";  
    volatile unsigned int canary3 = 0x87654321;  
  
    printf("Password (Hint it is '%s'): ", root_password);  
  
    scanf("%s", user_password);  
  
    if (canary1 != 0x12345678 || canary2 != 0xDEADBEEF  
        || canary3 != 0x87654321) {  
        printf("Stack manipulation detected!\n");  
        return false;  
    }  
  
    return strcmp(root_password, user_password) == 0;  
}
```

Prozesslayout: Stack im Detail



```
bool check_password() {
    volatile unsigned int canary1 = 0x12345678;
    volatile char root_password[] = "secret";
    volatile unsigned int canary2 = 0xDEADBEEF;
    volatile char user_password[16] = "0123456789ABCDE"; ◀
    volatile unsigned int canary3 = 0x87654321;

    printf("Password (Hint it is '%s'): ", root_password);

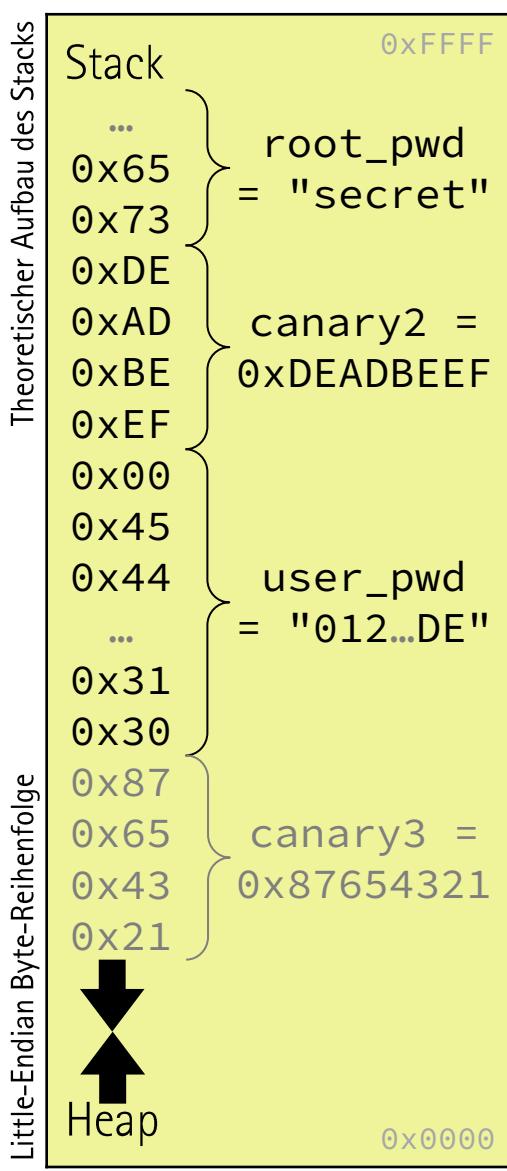
    scanf("%s", user_password);

    if (canary1 != 0x12345678 || canary2 != 0xDEADBEEF
        || canary3 != 0x87654321) {
        printf("Stack manipulation detected!\n");
        return false;
    }

    return strcmp(root_password, user_password) == 0;
}
```

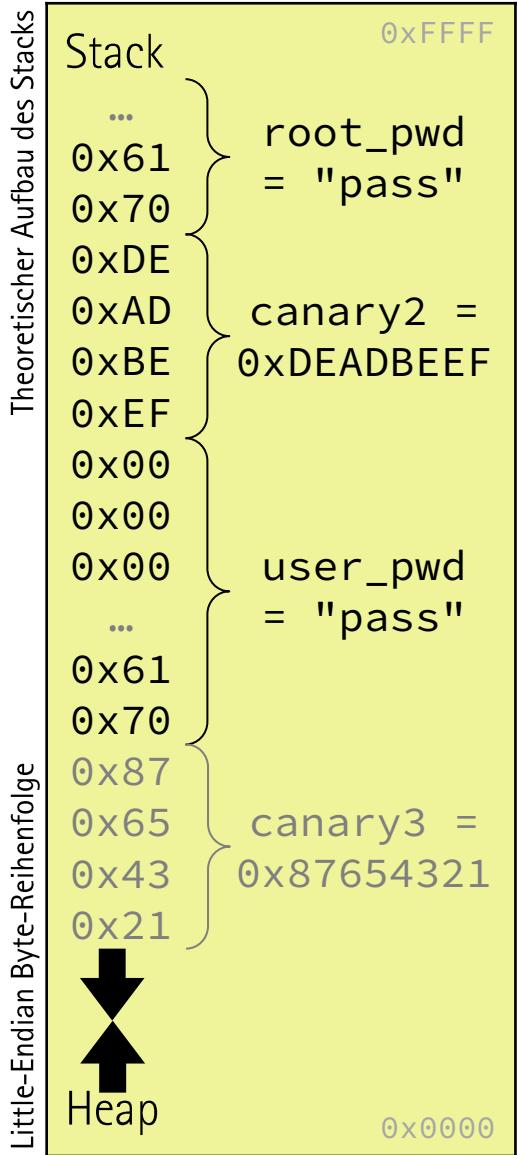
Prozesslayout: Stack im Detail

Theoretischer Aufbau des Stacks



```
bool check_password() {  
    volatile unsigned int canary1 = 0x12345678;  
    volatile char root_password[] = "secret";  
    volatile unsigned int canary2 = 0xDEADBEEF;  
    volatile char user_password[16] = "0123456789ABCDE";  
    volatile unsigned int canary3 = 0x87654321; // Red arrow points here  
  
    printf("Password (Hint it is '%s'): ", root_password);  
  
    scanf("%s", user_password);  
  
    if (canary1 != 0x12345678 || canary2 != 0xDEADBEEF  
        || canary3 != 0x87654321) {  
        printf("Stack manipulation detected!\n");  
        return false;  
    }  
  
    return strcmp(root_password, user_password) == 0;  
}
```

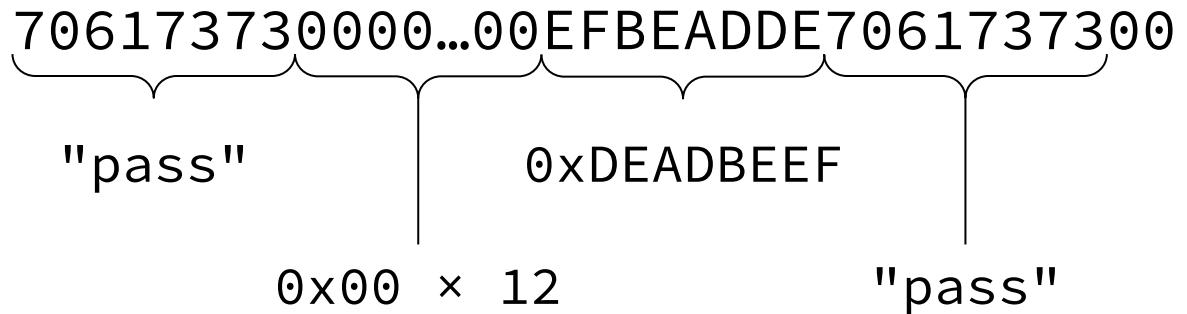
Prozesslayout: Stack-Overflow



```
bool check_password() {  
    volatile unsigned int canary1 = 0x12345678;  
    volatile char root_password[] = "secret";  
    volatile unsigned int canary2 = 0xDEADBEEF;  
    volatile char user_password[16] = "0123456789ABCDE";  
    volatile unsigned int canary3 = 0x87654321;  
  
    printf("Password (Hint it is '%s'): ", root_password);  
  
    scanf("%s", user_password);
```

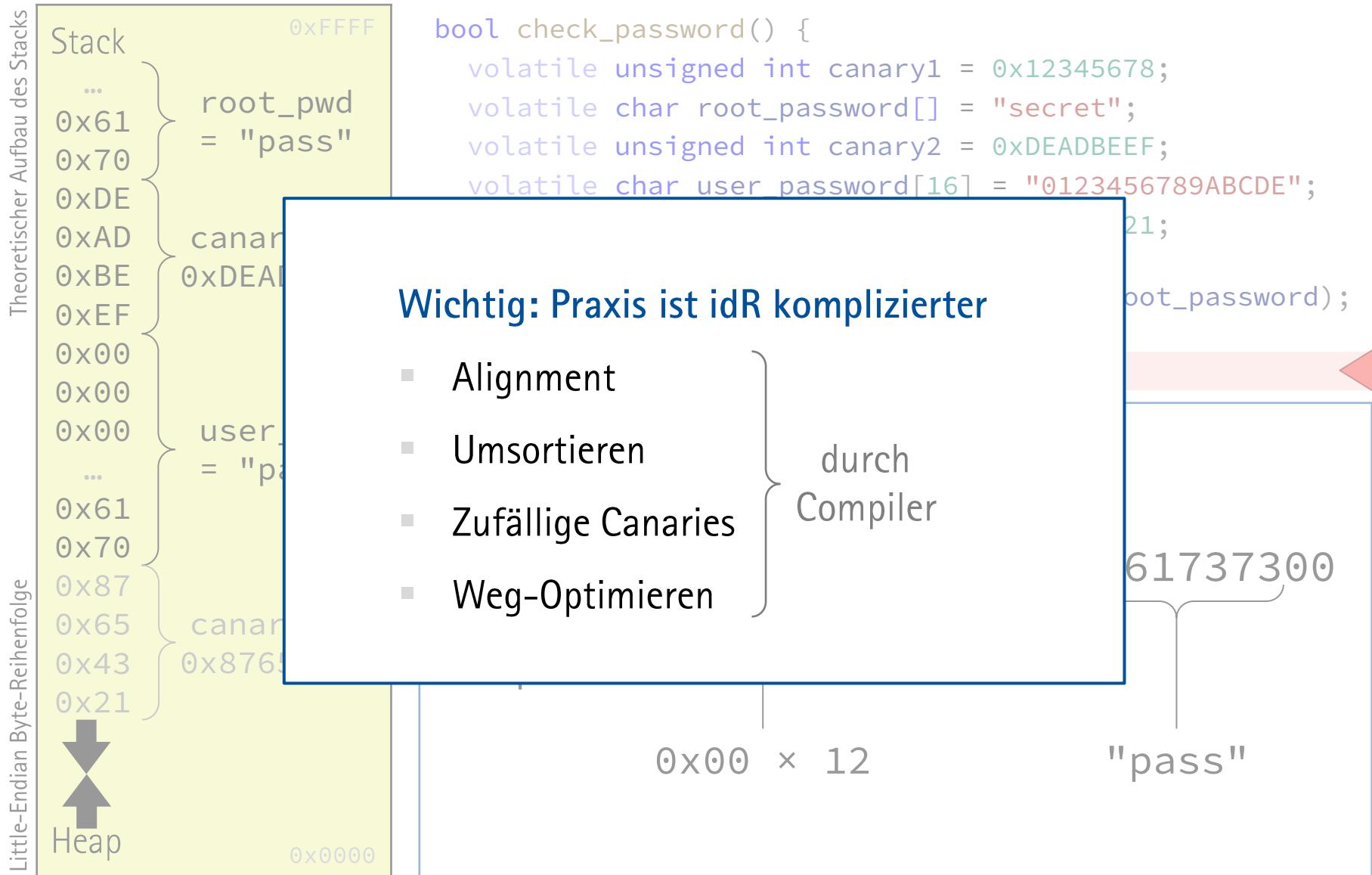
Mögliche Eingabe (binär):

706173730000...00EFBEADDE7061737300



"pass" 0x00 x 12 "pass"

Prozesslayout: Stack-Overflow



Auszug aus OpenSSH 3.3:

```
char **response;
unsigned int nresp = packet_get_int();
if (nresp > 0) {
    response = malloc(nresp * sizeof(char *));
}

for (int i = 0; i < nresp; i++)
    response[i] = packet_get_string(NULL);
}
```

Wertebereich:
0 bis 4.294.967.295

Auszug aus OpenSSH 3.3:

```
char **response;
unsigned int nresp = 1_073_741_825;
if (nresp > 0) {
    response = malloc(1_073_741_825 * 4);
    for (int i = 0; i < 1_073_741_825; i++)
        response[i] = packet_get_string(NULL);
}
```

Wertebereich:
0 bis 4.294.967.295

Angenommen:

- **nresp = 1_073_741_825**
- **sizeof(char *) = 4**

Auszug aus OpenSSH 3.3:

```
char **response;
unsigned int nresp = 1_073_741_825;
if (nresp > 0) {
    response = malloc(4_294_967_300);
    for (int i = 0; i < 1_073_741_825; i++)
        response[i] = packet_get_string(NULL);
}
```

Wertebereich:
0 bis 4.294.967.295

Angenommen:

- **nresp = 1_073_741_825**
- **sizeof(char *) = 4**

Auszug aus OpenSSH 3.3:

```
char **response;
unsigned int nresp = 1_073_741_825;
if (nresp > 0) {
    response = malloc(4);
    for (int i = 0; i < 1_073_741_825; i++)
        response[i] = packet_get_string(NULL);
}
```

Wertebereich:
0 bis 4.294.967.295

for-Schleife verletzt die

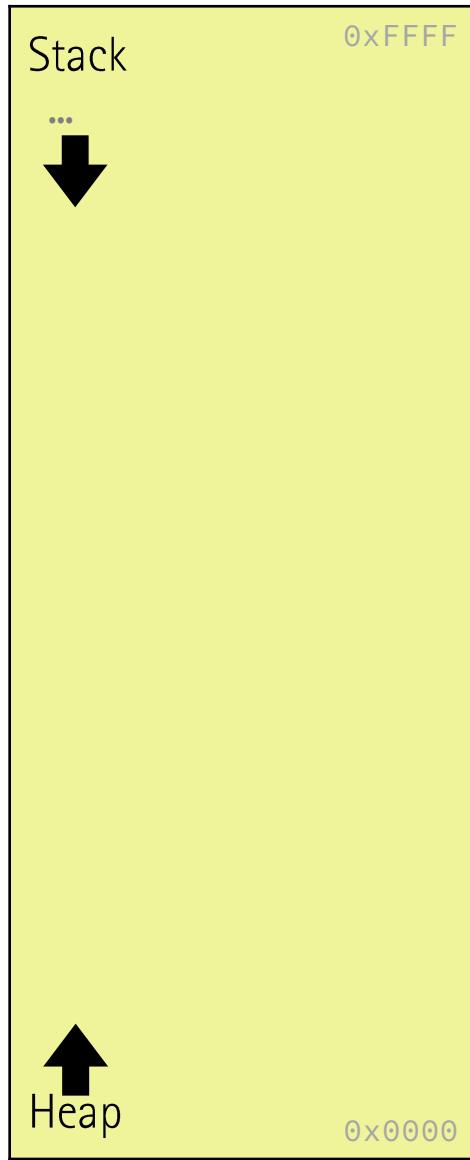
- *räumliche* Sicherheit (Buffer-Overflow)
- *zeitliche* Sicherheit (Speicher nicht allokiert)

Angenommen:

- **nresp** = **1_073_741_825**
- **sizeof(char *)** = **4**

Prozesslayout: Rücksprungadressen

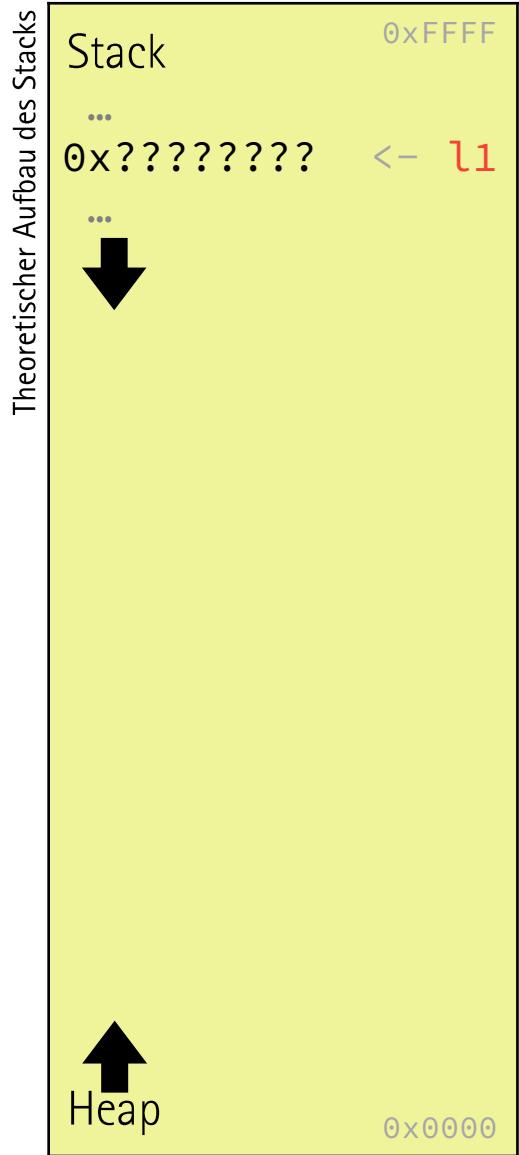
Theoretischer Aufbau des Stacks



```
int main() {  
    f1();  
    return true;  
}  
  
void f1() {  
    f2(0xBAADC0DE);  
    int a = 10;  
}  
  
void f2(unsigned int v) {  
    f3();  
    v += 5;  
}
```

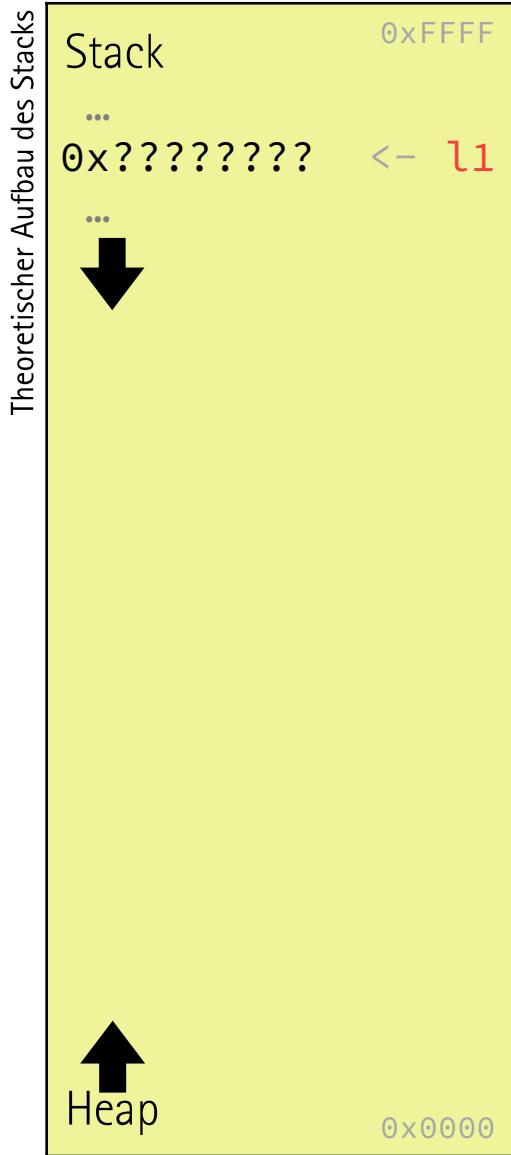


Prozesslayout: Rücksprungadressen



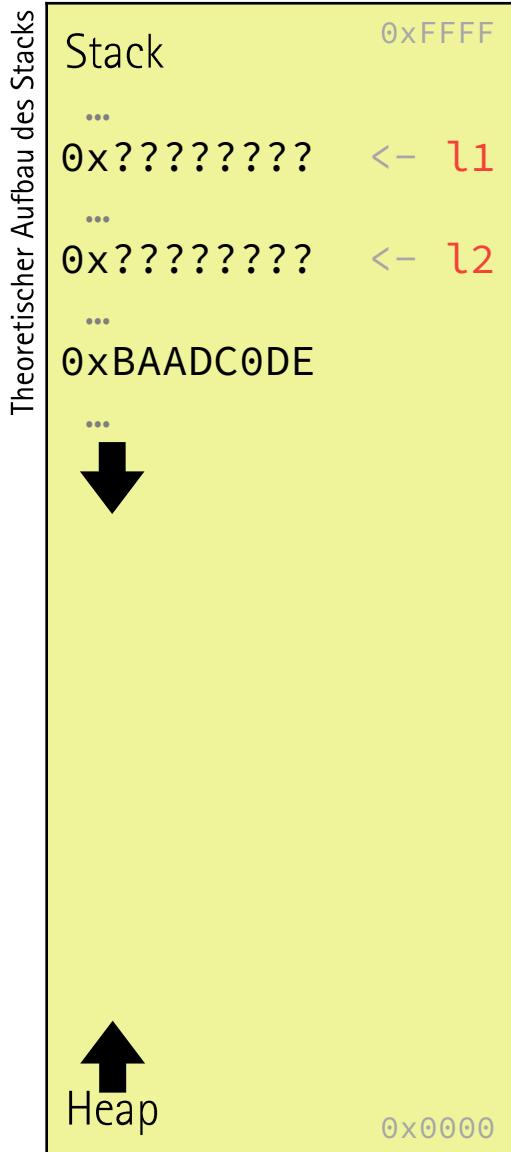
```
int main() {  
    f1();  
    return true;  
}  
  
void f1() {  
    f2(0xBAADC0DE);  
    int a = 10;  
}  
  
void f2(unsigned int v) {  
    f3();  
    v += 5;  
}
```

Prozesslayout: Rücksprungadressen

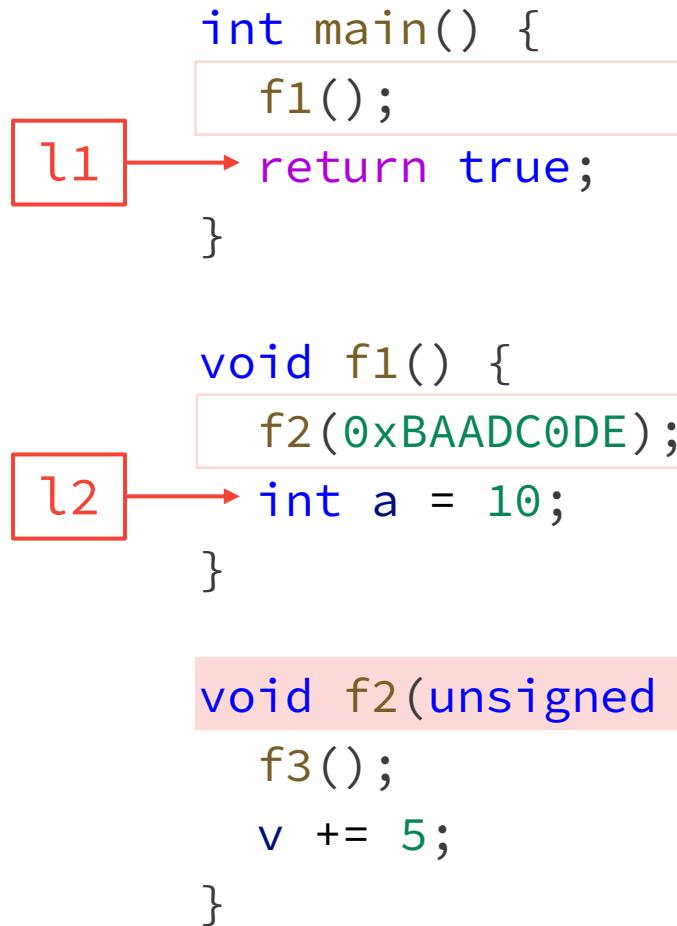


```
int main() {  
    f1();  
    return true;  
}  
  
void f1() {  
    f2(0xBAADC0DE);  
    int a = 10;  
}  
  
void f2(unsigned int v) {  
    f3();  
    v += 5;  
}
```

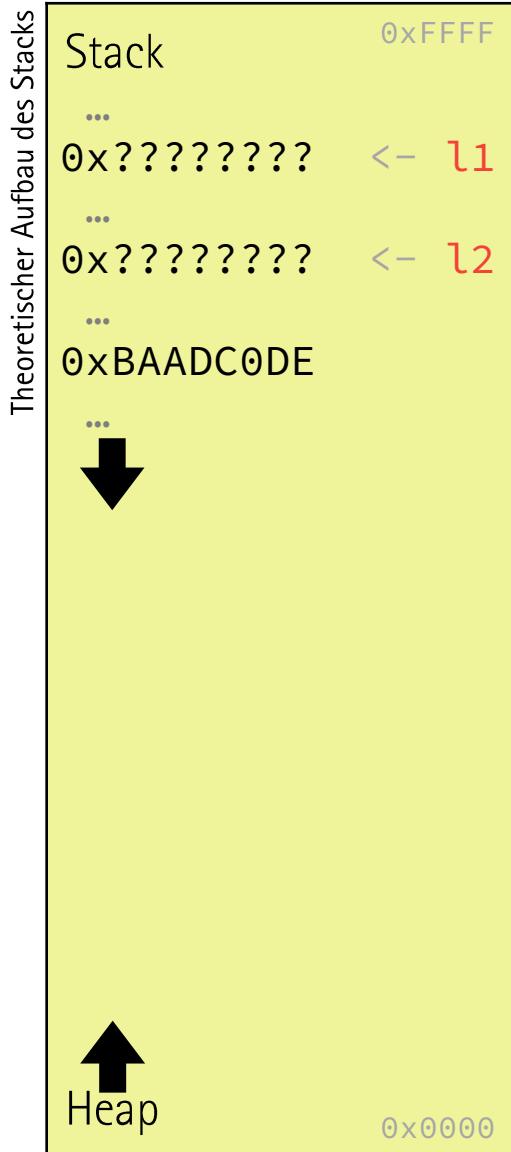
Prozesslayout: Rücksprungadressen



```
int main() {  
    f1();  
    return true;  
}  
  
void f1() {  
    f2(0xBAADC0DE);  
    int a = 10;  
}  
  
void f2(unsigned int v) {  
    f3();  
    v += 5;  
}
```

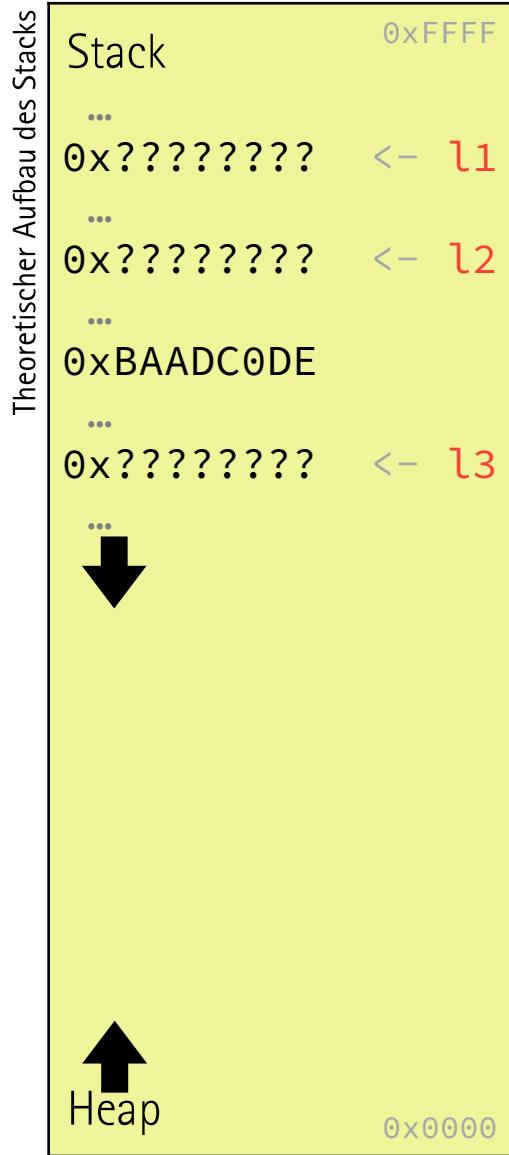


Prozesslayout: Rücksprungadressen

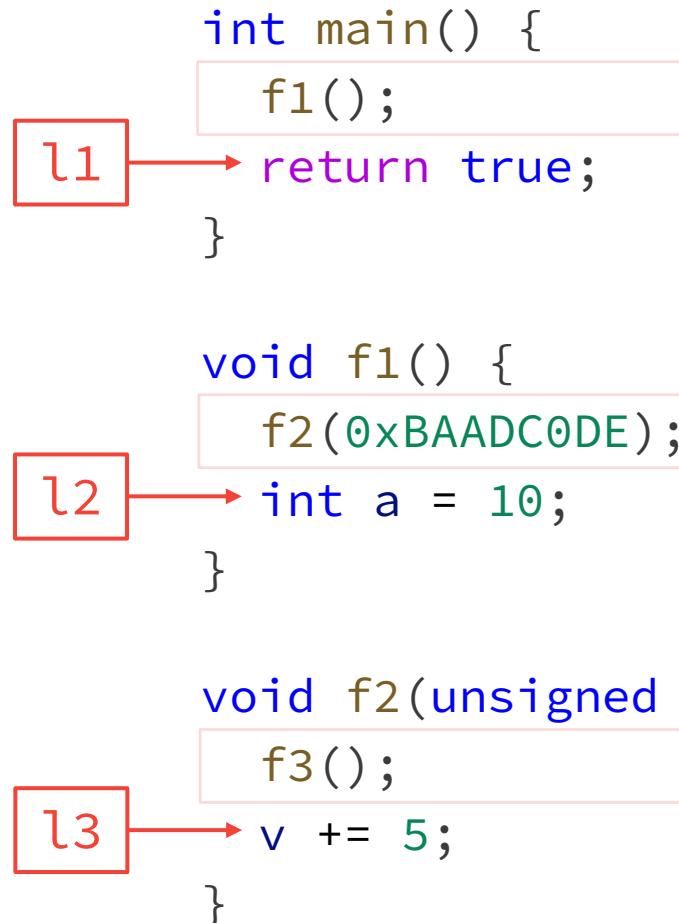


```
int main() {  
    f1();  
    return true;  
}  
  
void f1() {  
    f2(0xBAADC0DE);  
    int a = 10;  
}  
  
void f2(unsigned int v) {  
    f3();  
    v += 5;  
}
```

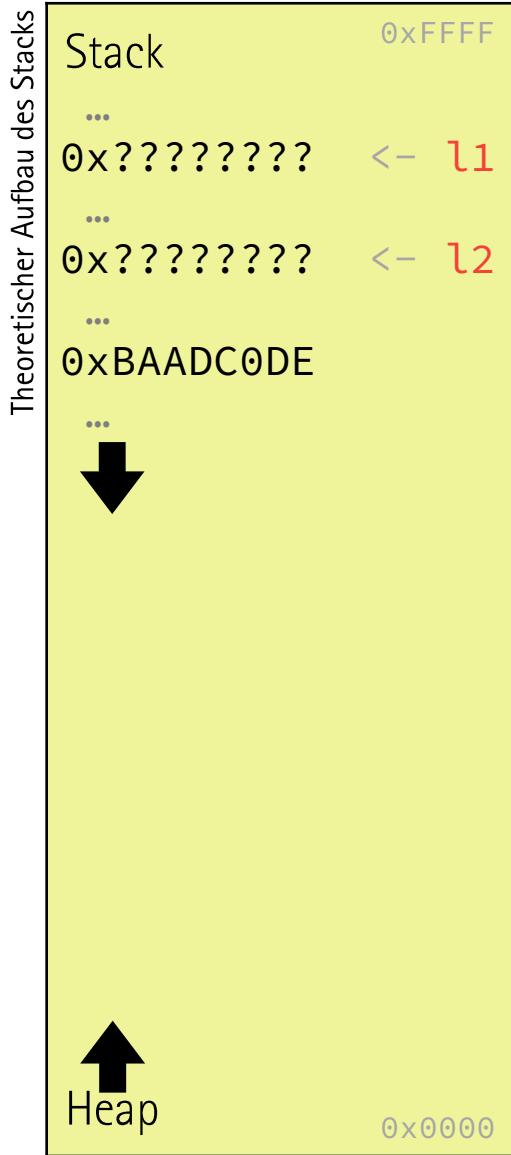
Prozesslayout: Rücksprungadressen



```
int main() {  
    f1();  
    return true;  
}  
  
void f1() {  
    f2(0xBAADC0DE);  
    int a = 10;  
}  
  
void f2(unsigned int v) {  
    f3();  
    v += 5;  
}
```



Prozesslayout: Rücksprungadressen



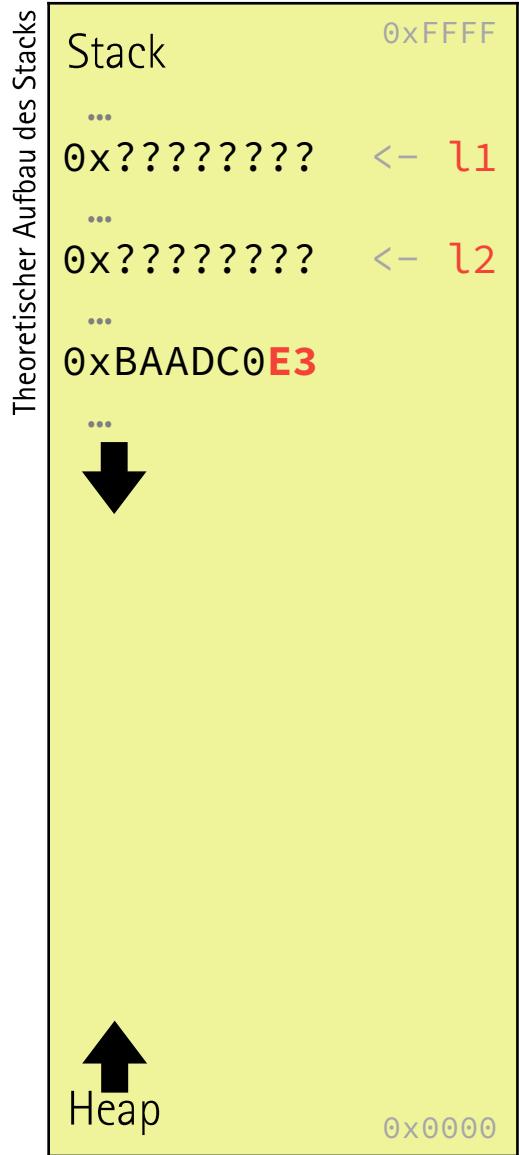
```
int main() {  
    f1();  
    return true;  
}  
  
void f1() {  
    f2(0xBAADC0DE);  
    int a = 10;  
}  
  
void f2(unsigned int v) {  
    f3();  
    v += 5;  
}
```

l1 → return true;

l2 → int a = 10;

l3 → v += 5;

Prozesslayout: Rücksprungadressen



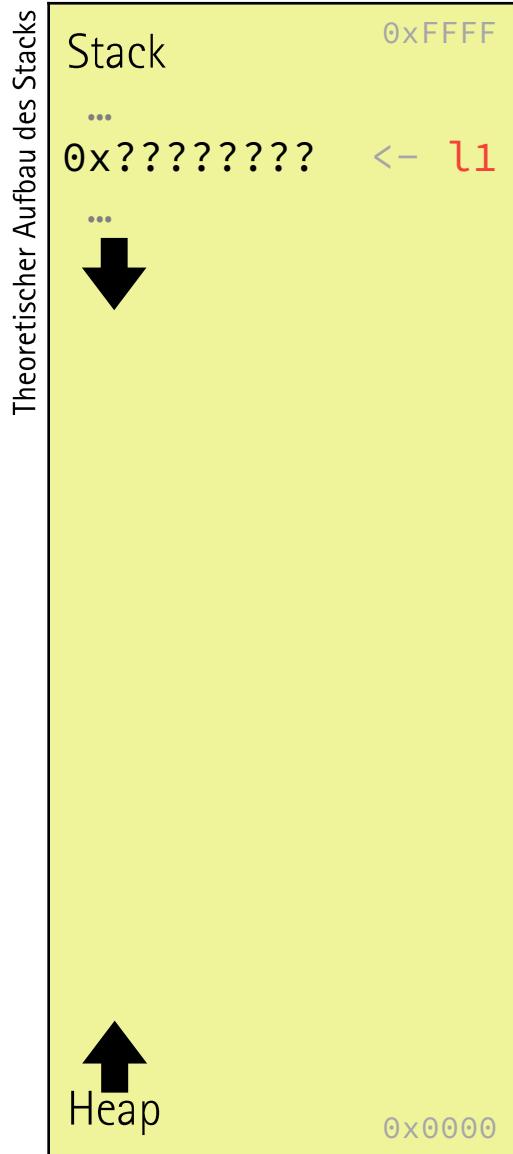
```
int main() {  
    f1();  
    return true;  
}  
  
void f1() {  
    f2(0xBAADC0DE);  
    int a = 10;  
}  
  
void f2(unsigned int v) {  
    f3();  
    v += 5;  
}
```

l1 → return true;

l2 → int a = 10;

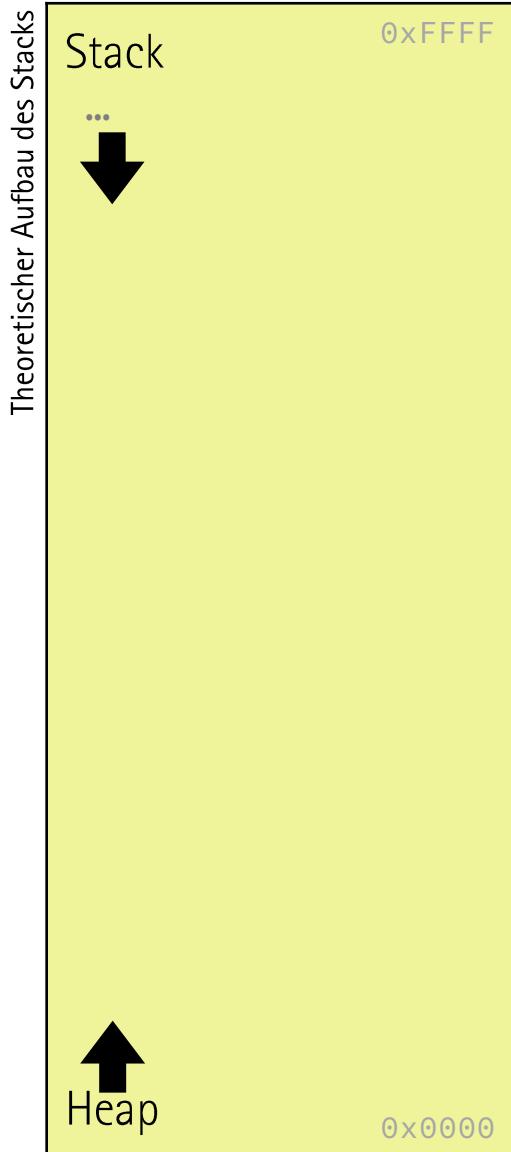
l3 → v += 5;

Prozesslayout: Rücksprungadressen



```
int main() {  
    f1();  
    return true;  
}  
  
void f1() {  
    f2(0xBAADC0DE);  
    int a = 10;  
}  
  
void f2(unsigned int v) {  
    f3();  
    v += 5;  
}
```

Prozesslayout: Rücksprungadressen



```
int main() {  
    f1();  
    l1 → return true;  
}  
  
void f1() {  
    f2(0xBAADC0DE);  
    l2 → int a = 10;  
}  
  
void f2(un  
    f3();  
    l3 → v += 5;  
}
```

Achtung:

Rücksprungadressen zeigen auf den Speicher, in dem die ausführbaren Maschinen-instruktionen liegen

→ Durch Stack-Overflow manipulieren der Rücksprungadresse möglich

Vorgehen zur Code-Ausführung

- Shellcode* in Buffer schreiben
- Rücksprungadresse auf diesen Buffer setzen

Beispiel: <https://cocomelonc.github.io/tutorial/2021/10/09/linux-shellcoding-1.html>

```
31 C0          xor eax, eax      ; zero out eax
31 DB          xor ebx, ebx      ; zero out ebx
31 C9          xor ecx, ecx      ; zero out ecx
31 D2          xor edx, edx      ; zero out edx
50             push eax         ; string terminator "\0"
68 6E 2F 73 68  push 0x68732f6e   ; "hs/n"
68 2F 2F 62 69  push 0x69622f2f   ; "ib//"
89 E3          mov ebx, esp       ; point ebx to "//bin/sh"
B0 0B          mov eax, 0xb        ; write 11 to eax (execve)
CD 80          int 0x80         ; syscall (interrupt 0x80)
```

} " //bin/sh"

*Shellcode = kompilierte Maschineninstruktionen „zum Starten einer Shell“

EXKURS: MELTDOWN UND SPECTRE

Oder auch: Wie können Seitenkanalangriffe funktionieren?

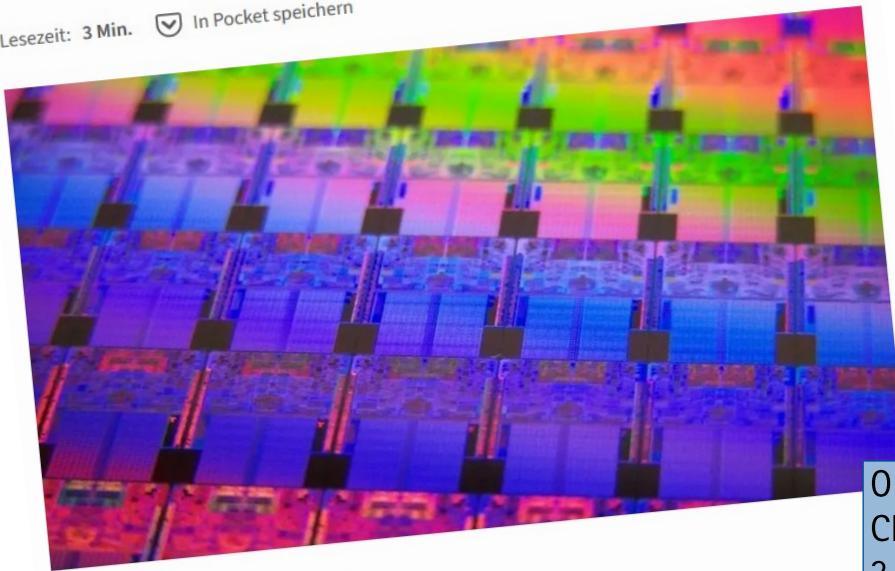
Meltdown und Spectre in der Presse

Massive Lücke in Intel-CPUs erfordert umfassende Patches

Derzeit arbeiten Linux- und Windows-Entwickler mit Hochdruck an umfangreichen Sicherheits-Patches, die Angriffe auf Kernel-Schwachstellen verhindern sollen. Grund für die Eile: eine Intel-spezifische Sicherheitslücke.



Lesezeit: 3 Min. In Pocket speichern



03.01.2018 13:51 Uhr | Security

Von Olivia von Westernhagen

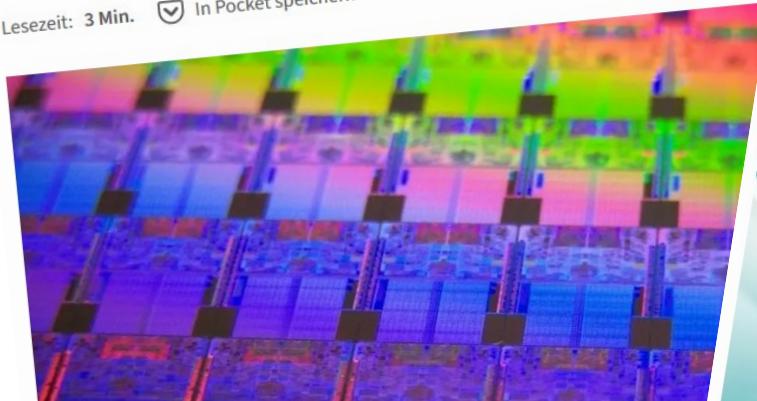
Olivia von Westernhagen, 2018. Massive Lücke in Intel-CPUs erfordert umfassende Patches. heise online [online]. 3. Januar 2018. Verfügbar unter: <https://www.heise.de/news/Massive-Luecke-in-Intel-CPUs-erfordert-umfassende-Patches-3931562.html>

Meltdown und Spectre in der Presse

Massive Lücke in Intel-CPUs erfordert umfassende Patches

Derzeit arbeiten Linux- und Windows-Entwickler mit Hochdruck an umfangreichen Sicherheits-Patches, die Angriffe auf Kernel-Schwachstellen verhindern sollen. Grund für die Eile: eine Intel-spezifische Sicherheitslücke

Lesezeit: 3 Min. In Pocket speichern



Gravierende Prozessor-Sicherheitslücke: Nicht nur Intel-CPUs betroffen, erste Details und Updates

Nach diversen Spekulationen über Ursache und Auswirkungen der CPU-Sicherheitslücke nehmen Intel und Google Stellung. Google veröffentlicht Details, außerdem zeigten Sicherheitsforscher mit Spectre und Meltdown zwei Angriffsszenarien.

Lesezeit: 6 Min. In Pocket speichern

1275



Jürgen Kuri, 2018. Gravierende Prozessor-Sicherheitslücke: Nicht nur Intel-CPUs betroffen, erste Details und Updates. heise online [online]. 4. Januar 2018. Verfügbar unter:
<https://www.heise.de/news/Gravierende-Prozessor-Sicherheitsluecke-Nicht-nur-Intel-CPUs-betroffen-erste-Details-und-Updates-3932573.html>

Meltdown und Spectre in der Presse

Kernel panic! What are Meltdown and Spectre, the bugs affecting nearly every computer and device?

Devin Coldewey @techcrunch / 2:47 AM GMT+1 • January 4, 2018

Mas
um
Derz
um
ve
U



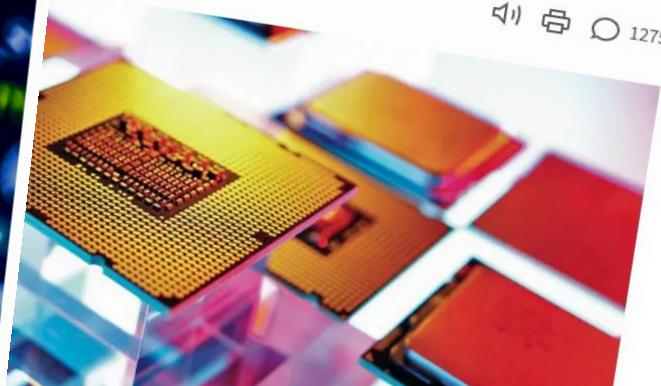
03.01.2018 13:51 Uhr | Security
Von Olivia von Westernhagen

Gravierende Prozessor-Sicherheitslücke: Nicht nur Intel ist betroffen, erste Details und

ationen über Ursache und Auswirkungen der CPU-nen Intel und Google Stellung. Google veröffentlicht Details, herheitsforscher mit Spectre und Meltdown zwei

cket speichern

1275



Devin Coldewey, 2018. Kernel panic! What are Meltdown and Spectre, the bugs affecting nearly every computer and device? TechCrunch [online]. 4. Januar 2018. Verfügbar unter: <https://techcrunch.com/2018/01/03/kernel-panic-what-are-meltdown-and-spectre-the-bugs-affecting-nearly-every-computer-and-device/>

Meltdown und Spectre in der Presse

Kernel panic! What are Meltdown and Spectre, the bugs affecting nearly every computer and device?

Devin Coldewey @techcrunch / 2:47 AM GMT+1 • January 4, 2018

Gravierende Prozessor-Sicherheitslücke: Nicht nur Intel-Cpus betroffen, erste Details und

Understanding Meltdown & Spectre: What To Know About New Exploits That Affect Virtually All CPUs

by Ryan Smith on January 4, 2018 11:45 AM EST

Posted in [CPUs](#) [AMD](#) [Intel](#) [Arm](#) [Spectre](#) [Security](#) [Meltdown](#)

210
Comments



Ryan Smith, 2018. Understanding Meltdown & Spectre: What To Know About New Exploits That Affect Virtually All CPUs. AnandTech [online]. 4. Januar 2018. Verfügbar unter: <https://www.anandtech.com/show/12214/understanding-meltdown-and-spectre>

Meltdown und Spectre in der Presse

Kernel panic! What are Meltdown and Spectre, the bugs affecting nearly every computer and device?

Devin Coldewey @techcrunch / 2:47 AM GMT+1 • January 4, 2018

Gravierende Prozessor-Sicherheitslücke: Nicht nur Intel-Cpus betroffen, erste Details und Patch

Understanding Meltdown & Spectre: What To Know About New Exploits That Affect Virtually All CPUs

2018 11:45 AM EST

Security Meltdown

“Meltdown” and “Spectre:” Every modern processor has unfixable security flaws

Immediate concern is for Intel chips, but everyone is at risk.

ARS STAFF - 1/4/2018, 1:30 AM

462

Windows, Linux, and macOS have all received [security patches](#) that significantly alter how the operating systems handle virtual memory in order to protect against a hitherto undisclosed flaw. This is more than a little notable; it has been clear that Microsoft and the Linux kernel developers have been informed of some non-public security issue and have been rushing to fix it. But nobody knew quite what the problem was, leading to lots of speculation and experimentation based on pre-releases of the patches.

Now we know what the flaw is. And it's not great news, because there are in fact [two related families of flaws](#) with similar impact, and only one of them has any easy fix.

The flaws have been named Meltdown and Spectre. Meltdown was independently discovered by three groups—researchers from the Technical University of Graz in Austria, German security firm Cerberus Security, and Google's Project Zero. Spectre was discovered independently by Project Zero and independent researcher Pa



Ars Staff, 2018. „Meltdown“ and „Spectre:“ Every modern processor has unfixable security flaws. Ars Technica [online]. 4. Januar 2018. Verfügbar unter:
<https://arstechnica.com/gadgets/2018/01/meltdown-and-spectre-every-modern-processor-has-unfixable-security-flaws/>

Meltdown und Spectre in der Presse

Kernel panic! What are Meltdown and Spectre, the bugs affecting nearly every computer an

Devin Coldewey @techcrunch / 2:47 AM GMT+1 • January 4, 2018

Mas
um
Derz
um



Understan
Know Ab
CPUs

“Meltdown” and “Spectre:” Every modern processor has unfixable security flaws

Immediate concern is for Intel chips, but everyone is at risk.

ARS STAFF - 1/4/2018, 1:30 AM

462

Windows, Linux, and macOS have all received [security patches](#) that significantly alter how the operating systems handle virtual memory in order to protect against a hitherto undisclosed flaw. This is more than a little notable; it has been clear that Microsoft and the Linux kernel developers have been informed of some non-public security issue and have been rushing to fix it. But nobody knew quite what the problem was, leading to lots of speculation and experimentation based on [pre-releases of the patches](#).

Andreas Stiller, 2018. Analyse zur Prozessorlücke: Meltdown und Spectre sind ein Security-Supergau. heise online [online]. 5. Januar 2018. Verfügbar unter: <https://www.heise.de/news/Analyse-zur-Prozessorluecke-Meltdown-und-Spectre-sind-ein-Security-Supergau-3935124.html>

Gravierende Prozessor-Sicherheitslücke: Nicht nur Intel, sondern auch Us betroffen. erste Daten Analyse zur Prozessorlücke: Meltdown und Spectre sind ein Security-Supergau

Die Sicherheitslücken Meltdown und Spectre treffen die Prozessorhersteller ins Mark - vor allem Intel. Aus den Lücken ergeben sich mehr als ein Dutzend Angriffsmöglichkeiten - ein Security-Supergau. Eine Analyse von Andreas Stiller.

Lesezeit: 8 Min. In Pocket speichern

1188



(Bild: Alysson alyssonrock, gemeinfrei (Creative Commons CC0))

05.01.2018 19:33 Uhr | Security
Von Andreas Stiller

Meltdown und Spectre in der Presse

Kernel panic ...
Gravieren ...
Aussehen vor Sicherheit
Von [REDACTED]

Seit zwei Jahrzehnten sind in unseren Computern, Laptops und Handys Chips mit gravierenden Sicherheitslücken verbaut. Anstatt das Problem ein für alle Mal zu beheben und die Prozessoren auszutauschen – was natürlich mit einem hohen Aufwand in Verbindung stehen würde –, versuchen Firmen lediglich, die Lücke mit Software-Updates zu stopfen. Denn: Das Design der Chips müsste für die Behebung verändert werden. Für den Designer undenkbar. Das Aussehen geht scheinbar vor Sicherheit. Stattdessen sind die Daten von Tausenden Millionen Menschen für Hacker mit einer Leichtigkeit zu bekommen. Was bisher über die Sicherheitslücke bekannt ist, lesen Sie auf der Wirtschaftsseite.

462

Win that virtu und clea info rush lead pre-
Now beca simi
The was

“Meltdo process Immediate conce ARS STAFF - 1/4/2018, 1:30

Enlarge / Spectre Creative Commons CC0) 05.01.2018 19:33 Uhr | Security Von Andreas Stiller

Nicht und ersteller ins zendreas Stiller. 1188

Meltdown und Spectre

- Verwandte Seitenkanalangriffe vom Januar 2018
 - [CVE-2017-5754](#) (Meltdown)
 - [CVE-2017-5753](#) (Spectre V1)
 - [CVE-2017-5715](#) (Spectre V2)
- Grundlage: Spekulative Ausführung und Sprungvorhersage
- Theoretisch können Prozesse beliebigen Arbeitsspeicher des Kernels auslesen
- **Keine Vertraulichkeit mehr**

CVSS Score
5.6 MEDIUM

CVSS:3.1/AV:L/AC:H/PR:L/UI:N/S:C/C:H/I:N/A:N

Attack Vector:	Local
Attack Complexity:	High
Privileges Required:	Low
User Interaction:	None
Scope:	Changed
Confidentiality Impact:	High
Integrity Impact:	None
Availability Impact:	None



Einschub: Seitenkanalangriffe

```
int main() {  
    // ...  
    char pwd[1024];  
    scanf("%s", pwd);  
    check_password(pwd, "secret");  
    // ...  
}
```

Probleme?

```
bool check_password(char* pwd1, char* pwd2) {  
    if (strlen(pwd1) != strlen(pwd2))  
        return false;  
    for (int i = 0; i < strlen(pwd1); i++)  
        if (pwd1[i] != pwd2[i])  
            return false;  
    return true;  
}
```

Passwort kann zeichenweise erraten werden.
Komplexität sinkt von $|A|^n$ auf $n \cdot |A|$.

Hochsprachenebene

Assemblersprachenebene

Maschinenprogrammebene

Befehlssatzebene

MikroarchitekturEbene

Logikebene

Horizontale Isolation

(Anwendung/Anwendung)

- Interrupts & Memory-Mapping
- Realer vs. Logischer Adressraum

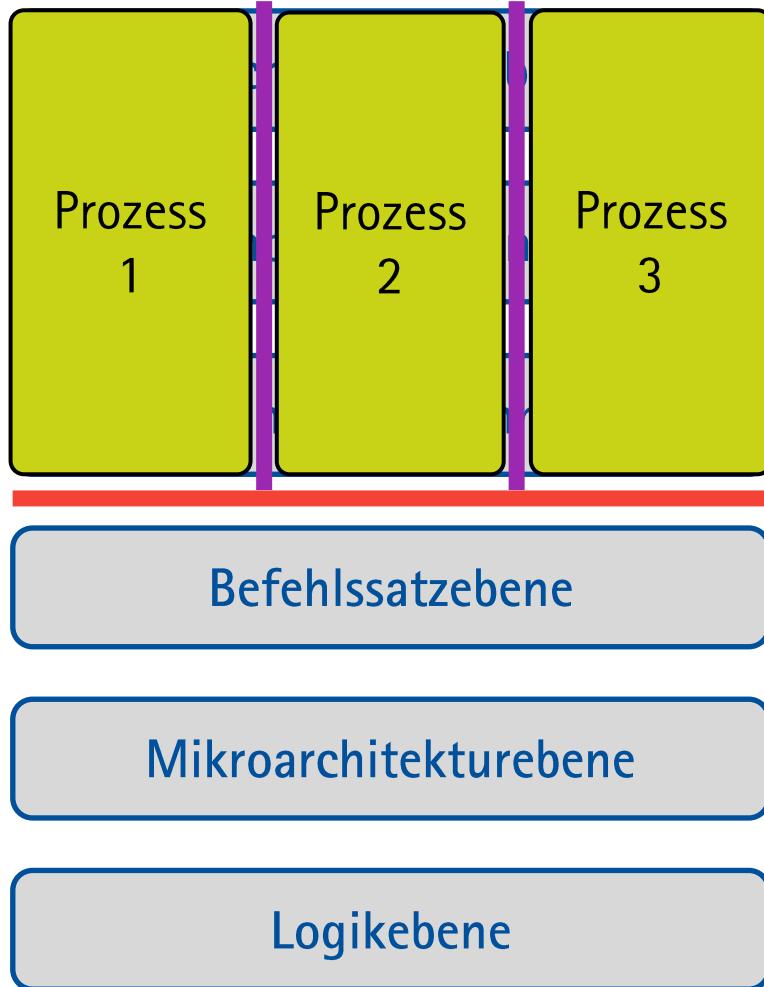
← Betriebssystem

Vertikale Isolation

(Benutzer/System)

- Privilegierte Operationen

Einschub: Speicherisolation



Horizontale Isolation

(Anwendung/Anwendung)

- Interrupts & Memory-Mapping
- Realer vs. Logischer Adressraum

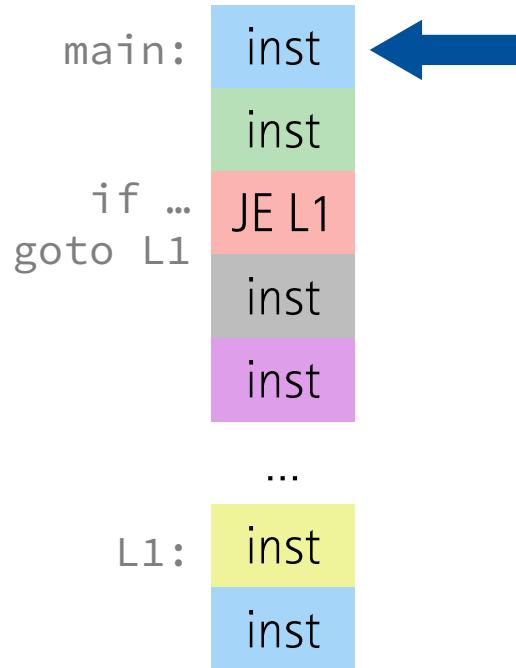
← Betriebssystem

Vertikale Isolation

(Benutzer/System)

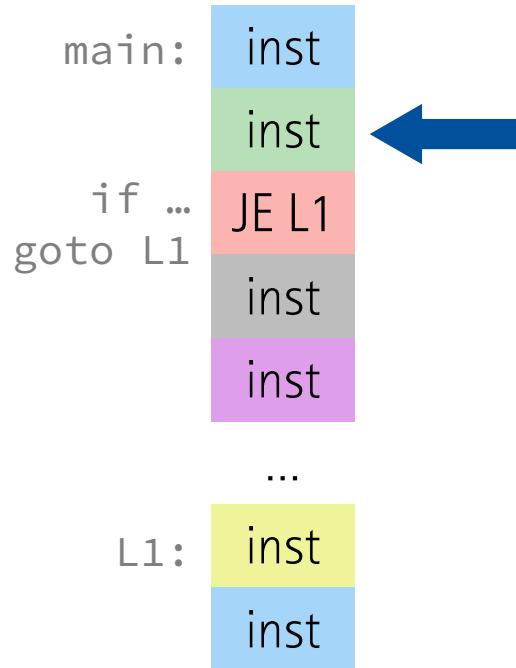
- Privilegierte Operationen

Einschub: Spekulative Ausführung



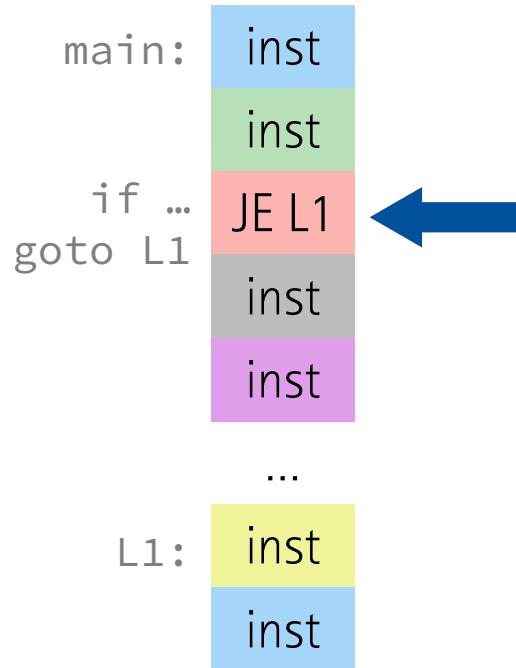
#	IF	ID	EX	WB
1	inst			
2				
3				
4				
5				
6				
7				

Einschub: Spekulative Ausführung



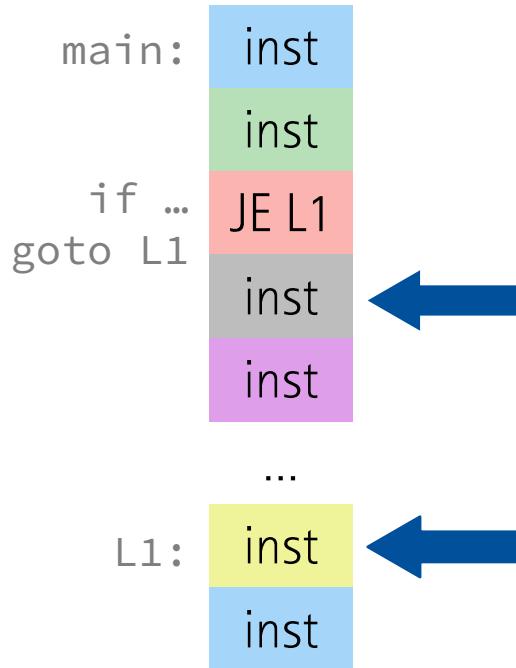
#	IF	ID	EX	WB
1	inst			
2	inst	inst		
3				
4				
5				
6				
7				

Einschub: Spekulative Ausführung



#	IF	ID	EX	WB
1	inst			
2	inst	inst		
3	JE L1	inst	inst	
4				
5				
6				
7				

Einschub: Spekulative Ausführung

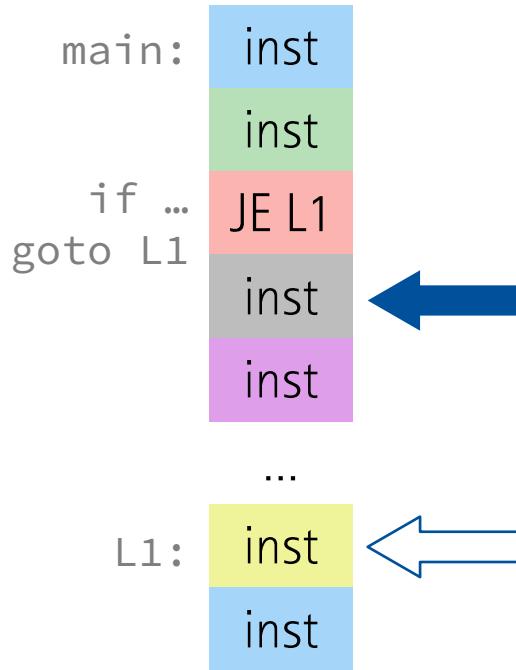


#	IF	ID	EX	WB
1	inst			
2	inst	inst		
3	JE L1	inst	inst	
4				
5				
6				
7				

Sprungziel unklar?

- Raten, Ausführen und ggf. Verwerfen

Einschub: Spekulative Ausführung

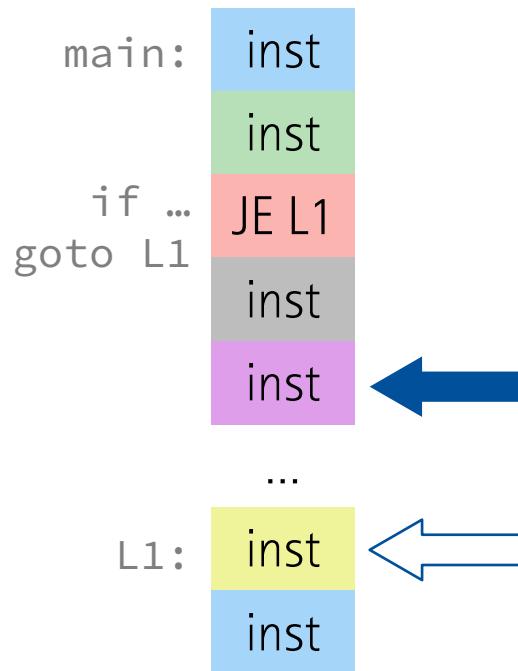


#	IF	ID	EX	WB
1	inst			
2	inst	inst		
3	JE L1	inst	inst	
4	inst	JE L1	inst	inst
5				
6				
7				

Sprungziel unklar?

- Raten, Ausführen und ggf. Verwerfen

Einschub: Spekulative Ausführung

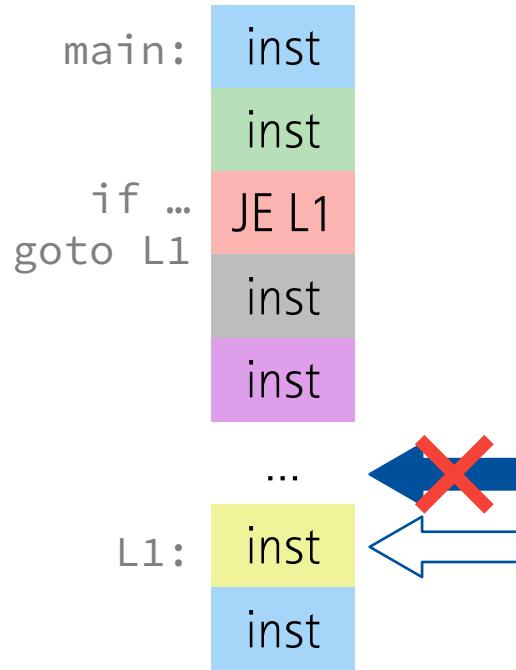


#	IF	ID	EX	WB
1	inst			
2	inst	inst		
3	JE L1	inst	inst	
4	inst	JE L1	inst	inst
5	inst	inst	JE L1	inst
6				
7				

Sprungziel unklar?

- Raten, Ausführen und ggf. Verwerfen

Einschub: Spekulative Ausführung

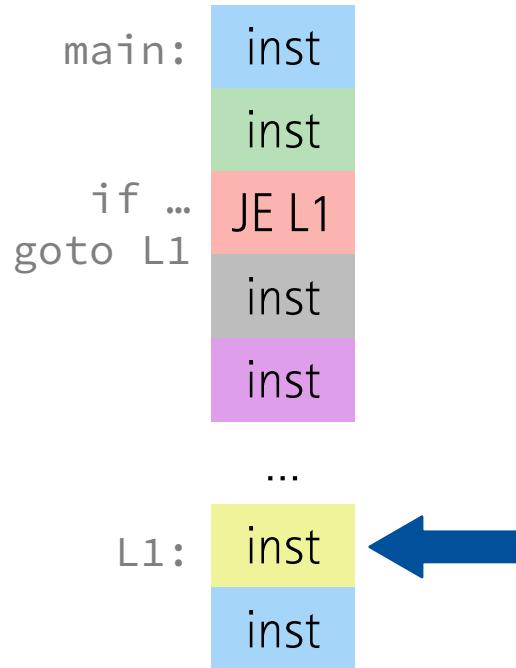


#	IF	ID	EX	WB
1	inst			
2	inst	inst		
3	JE L1	inst	inst	
4	inst	JE L1	inst	inst
5	inst	inst	JE L1	inst
6	...	inst	inst	JE L1
7				

Sprungziel unklar?

- Raten, Ausführen und ggf. Verwerfen

Einschub: Spekulative Ausführung



#	IF	ID	EX	WB
1	inst			
2	inst	inst		
3	JE L1	inst	inst	
4	inst	JE L1	inst	inst
5	inst	inst	JE L1	inst
6	...	inst	inst	JE L1
7	inst			

Sprungziel unklar?

- Raten, Ausführen und ggf. Verwerfen

Ähnliches gilt für (unberechtigte) Speicherzugriffe

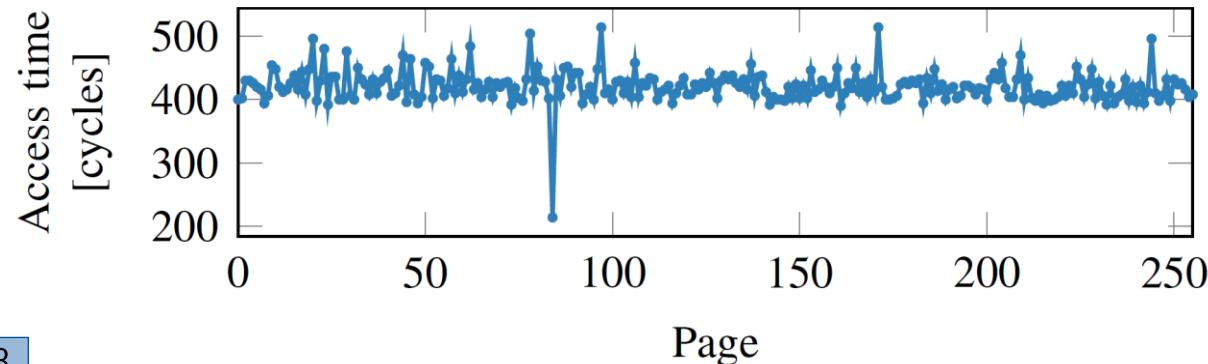
Meltdown im Detail

```
char array[256 * 4096];
char *hidden = ...;                                // Verbotene Adresse im Kernel
char secret = *hidden;                            // Lese verbotene Adresse
char dummy = array[secret * 4096];                // Verwende Geheimnis indirekt

// Ausnahme abfangen und behandeln

for (int i = 0; i < 256; ++i) {
    time( array[ i * 4096 ] );                  // Messe Zugriffszeiten
}
```

Seitenkanal =
Cachepräsenz



M. Lipp et al., 'Meltdown', arXiv. 2018.
DOI 10.48550/arXiv.1801.01207

Wenn Sie sich eine Sache von heute merken...

Eine Schwachstelle, die "zu kompliziert ist, als dass man sie jemals finden könnte", wird gefunden werden!

Heute geht es weiter mit Quiz 4!

Bearbeitungszeitraum:
Montag, 11.12.2023 18:00 – Montag, 18.12.2023 16:00

EVALUATION DER HEUTIGEN ÜBUNG

... oder über Stud.IP im Ordner der Vorlesung

