

Buffer Overflows

Praktische Analyse von Schwachstellen

Jakob Stühn, John Meyerhoff, Sam Taheri

H-BRS

January 10, 2022

Inhalt

- Geschichte
- Grundlegende Theorie
 - Speicheraufbau
 - Stack-/Heap-Overflow
- Shellcode
- Praktische Analyse
 - Programmierfehler
 - Demonstration
- Gegenmaßnahmen
- Fazit

Geschichte: Bekannte Buffer Overflows

- The Morris Worm (November 1988)
- SQL-Slammer (Januar 2003)
- HP-Drucker Firmware (2018)
- WhatsApp MP4 (2019)

Grundlegende Theorie

Definition

Im weitesten Sinne beschreibt ein Buffer Overflow eine Schwachstelle in einem Computerprogramm, bei der ein Angreifer einen Speicherbereich fester Größe überschreibt und diesen so zum “Überlaufen” bringt. Durch Ausspähen und Analysieren der Software kann dieses Überschreiben so gezielt geschehen, dass der Fluss des Programms verändert und zuvor injizierter Schadcode ausgeführt wird.

Grundlegende Theorie

Speicheraufbau

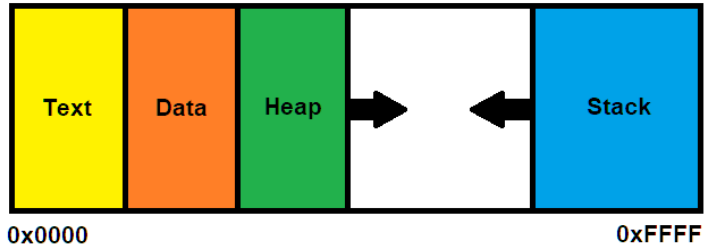


Figure: Prozess im Speicher

Grundlegende Theorie

Stack Overflow

- Wert einer Variable verändern
- Function Pointer manipulieren
- Return Pointer überschreiben



Figure: Buffer im Stack während eines Overflows

Grundlegende Theorie

Heap Overflow

Während der Laufzeit eines Programms allozierter Speicher (z. B. durch `malloc()`) wird im Heap angelegt. Dabei setzt sich jeder Speicherblock aus einem Header und dem tatsächlich angeforderten Speicher zusammen. Der Header enthält hierbei, je nach Implementation, Informationen über den Block, wie z. B. seine Größe. Aus diesen Informationen kann dann abgeleitet werden, an welcher Stelle der nächste Block beginnt.

Shellcode

Definition

- Shellcode ist eine Folge von Anweisungen.
Diese kann über einen Exploit in einen Prozess injiziert werden.
- In der Computersicherheit meint Shellcode ursprünglich Code, der bei der Ausführung eine Shell öffnet
- Üblicherweise in Assembly

Shellcode

Beispiel

6a 42	push 0x42
58	pop rax
fe c4	inc ah
48 99	cqo
52	push rdx
48 bf 2f 62 69 6e 2f	movabs rdi, 0x68732f2f6e69622f
2f 73 68	
57	push rdi
54	push rsp
5e	pop rsi
49 89 d0	mov r8, rdx
49 89 d2	mov r10, rdx
0f 05	syscall

Shellcode

Register zum Zeitpunkt des Syscalls

Die verwendeten Register:

RAX: 322 (Nummer des Syscalls)

RDI: 0x68732f2f6e69622f (Pfad der auszuführenden Datei: `"/bin//sh"`)

RSI: Pointer auf RDI (Zeiger auf den Pfad)

Die optionalen bzw. nicht verwendeten Register:

RDX: 0 (Optional)

R10: 0 (Optional)

R8: 0 (Optional)

R9: ? (Nicht verwendet)

Praktische Analyse

Programmierfehler

Overflow-anfällig sind Sprachen, welche direkte Zugriffe auf die Speicherstrukturen des Systems ermöglichen:

- Assembly
- C/C++
- Fortran

Praktische Analyse

Programmierfehler

Problematisch sind Funktionen, welche keine Kontrolle auf die Länge des Inputs implementieren:

- `gets(buffer)`
Erwartet Input und kopiert diesen in den angegebenen Speicher
- `strcpy(buffer, input)`
Kopiert einen Input in den angegebenen Speicher

Praktische Analyse

Format-String-Schwachstelle

```
printf('%s', chars)
```

Ist eine gute Umsetzung,

```
printf('%s', chars)
```

hingegen nicht.

Gegenmaßnahmen

Struktur

- Stack-Schutz mit “Canary” (Zufallszahl)
- Safe Pointer Instrumentalisierung
- C Range Error Detector und Out Of Bounds Object
- Hardware-basierte Lösungen
- Statische Code-Analyse
- Betriebssystembasierte Ansätze
- Manuelles Buffer-Overflow Blocken (Input-Bereinigung)

Gegenmaßnahmen

Code-Beispiel

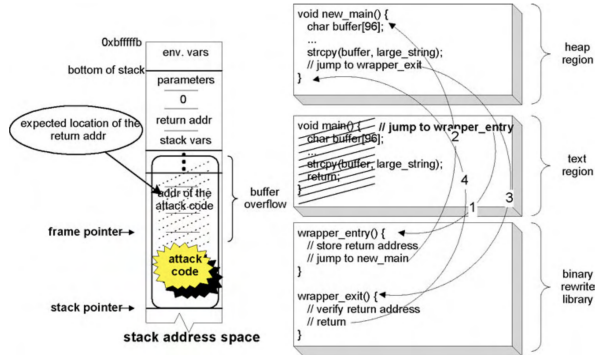


Figure 9: Libverify function call and stack layout

Gegenmaßnahmen

Testen

- Fuzzy Tests
- Spezifische Payloads