Versuch A: Buffer Overflow

Lara Quitte, Tino Jeromin

Stichwörter

Die folgenden Begriffe sind notwendig für den Versuch zu definieren:

- Format String Attack
- printf( ) (die C-Funktion)

Diese, sowie weitere wichtige Begriffe, sollen im Folgenden kurz definiert werden.

Format String Attack
Format String Attacks bezeichen eine Sicherheitslücke in der Programmierung, bei der Angreifer ein Programm zum Absturz bringen, aber auch potentiell schädlichen Code auszuführen können. Es handelt sich hierbei nicht um Sicherheitslücken im klassischen Sinne, viel eher ist es bei Format String Attacks möglich sich Bugs im Code zu Nutze zu machen.

printf( )
Die printf( )- Funktion ist eine Ausgabefunktion, welche ihren Ursprung in der Programmiersprache C hat. Die Funktion gehört zu den Format Funktionen, welche eine variable Anzahl von Argumenten als Eingabewerte entgegen nehmen. Einer dieser Werte ist der sogenannte Format String. Dieser ist ein ASCIIZ String, welcher sowohl Text als auch Format Parameter enthält, welche wiederum in der Ausgabe mit einem Wert ersetzt wird. Der Format Parameter gibt an, um welche Art von Ausgabe es sich handelt, beispielsweise Dezimalzahlen oder auch Strings.

Format Funktion
Eine Format Funktion ist eine ANSI C Funktion, wie *printf( )*, die eine primitive Variable der Programmiersprache in eine, vom Menschen lesbare, String Repräsentation konvertiert.

Format String
Ein Format String ist ein Argument der Format Funktion und besteht aus einem ASCIIZ String, welcher sowohl Text, als auch Format Parameter enthält.

Format String Parameter
Ein Format String Parameter definieren den Typ der Umwandlung in einer Format Funktion.

Teil 2: Format String Attack

Fragestellung: Wie funktioniert eine Format String Attack grundsätzlich?
Bei der Format String Attack macht sich der Angreifer eine fehlerhafte Implementation der *printf( )-Funktion* zu Nutze. Wird in dieser kein Format String angegeben, so ist es möglich diese Schwachstelle auszunutzen, indem man einen String übergibt, der gültige **Format-Parameter** enthält. Diese werden dann als *Format String* interpretiert und ausgeführt, wodurch vom Speicher gelesen wird, was der Angreifer schadhaft ausnutzen kann. Die *Format Funktion* ließt dann einfach die nächsten Daten des Speichers, wodurch es zum Kontrollverlust über den Prozess kommen kann.

Um den zweiten Teil des Versuches durchzuführen, haben wir folgende Schritte durchgeführt:

1. Herunterladen des C-Programms fmtstr.c

2. Kompilieren des C-Programms fmtstr.c mittels gcc → führt zu zwei Warnings, aber zu keinem Error:

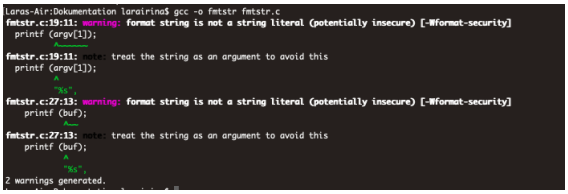

Figure 1: Kompilieren des Quellcodes

3. Einfaches Ausführen des Programms ohne Kommandozeilenargumente führt zu folgender Ausgabe:

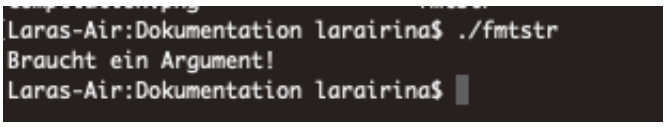

Figure 2: Ausführen des Quellcodes

4. Durch Eingabe zur Laufzeit des unveränderten Programms soll der Wert der Variable **x** verändert werden → Eingabe: "%x %x %x %x %x %x %x %x %n"

5. Das Vorzeichen der Variable **x** soll nun durch eine weitere Eingabe umgedreht werden → Eingabe "%155x %9$n"

Durch Veränderung der Print-Anweisung in Zeile 30 (siehe Figure 3) ist es einem Angreifer nun nicht mehr möglich dem Compiler einen eigenen Format String zu übergeben und ein Angriff wird so verhindert, da die Variable **buf** nun nur noch als String, also als Daten-Parameter, interpretiert wird.

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4
5  int
6  main (int argc, char *argv[])
7  {
8    char buf[128];
9    char x;
10   char *px = &x;
11
12   *px = 100;
13
14   if (argc != 2) {
15     printf ("Braucht ein Argument!\n");
16     exit (1);
17   }
18
19   printf (argv[1]);
20   putchar('\n');
21
22   printf ("x = %d\n", x);
23   printf ("Eingabe: ");
24   fflush (stdout);
25
26   if (fgets (buf, sizeof buf, stdin))
27   /*durch Einfügen des Format Strings %s wird die Variable buf nur noch als String interpretiert
28   * und der Angreifer hat nicht mehr die Möglichkeit den Code durch Eingabe eines eigenen Format Strings
29   * zu manipulieren. */
30     printf ("%s" , buf);
31
32   printf ("x = %d\n", x);
33   return 0;
34 }
```

Figure 3: Verbesserter Quellcode

Fragen

Wie kann ein*e Programmierer*in einfach dafür Sorge tragen, dass Format String Angriffe nicht mehr möglich (oder zumindest erheblich erschwert) sind?
Eine einfache Möglichkeit Format String Angriffe zu verhindern oder erschweren ist die Analyse des Programmcodes. Wird einer Format Funktion wie *printf( )* nur ein Wert übergeben, so kann man davon ausgehen, dass hier ein Angriff möglich ist. Schwachstellen werden so effizient vermieden und die Analyse lässt sich auch automatisiert ausführen. Trotz aller Einfachheit ist hierbei zu beachten, dass ausreichende Programmierkenntnisse notwendig sind um Patches zu erstellen und Quellcodes zur manuellen Analyse häufig zu komplex sind.

Eine weitere Möglichkeit zur Prävention ist das Prüfen auf Speichergrenzen von Variablen während der Programmausführung. So können Buffer Overflows während der Laufzeit erkannt und Angriffe vermieden werden. Um diese Maßnahme anwenden zu können, muss der Quellcode jedoch überarbeitet werden und auch die Performanz kann darunter leiden.

Wie lässt sich einfach herausfinden, ob ein derartiger Angriff vermutlich möglich ist?
Durch eine Analyse des Quelltextes lässt sich einfach herausfinden, ob ein Format String Angriff möglich ist. Hierbei gilt es herauszufinden, wie viele Werte einer Funktion der printf( )-Familie übergeben werden. Bekommt die Funktion nur einen Wert übergeben, so wird dies ials eine Art benutzerdefiniertes Format interpretiert. Werden jedoch mehr als ein Wert übergeben, so ist davon auszugehen, dass die Format Strings fest vorgegeben sind und ein derartiger Angriff nicht möglich ist.