AddressSanitizer (ASAN)

Beim AddressSanitizer handelt es sich um eine Compiler-Option (u. a. für clang), die Programmierern dabei helfen soll, Programmier-Fehler, die während der Laufzeit auftreten können, zu finden und zu beheben. Viele dieser Fehler können einen kritischen Einfluss auf das Programm haben, nicht nur in Bezug auf Stabilität, sondern insbesondere auch auf die Sicherheit, da durch solche Fehler häufig Sicherheitslücken entstehen. Deshalb ist die Verwendung von ASAN während der Entwicklung ratsam, da so Fehler ggf. während des Testens gefunden werden können.

HINWEIS: Bei der Verwendung von ASAN wird die Laufzeit eines Programmes extrem verlangsamt. Deshalb sollte ASAN nur während der Entwicklung verwendet werden, nicht aber für veröffentlichte Versionen.

Um eine Quellcode-Datei mit ASAN zu kompilieren, ist lediglich die Compiler-Flag „-fsanitize=address“ notwendig, z. B. „clang –o main main.c –fsanitize=address“. Die erzeugte ausführbare Datei ist nach außen von der Funktionalität her unverändert, beim Ausführen verrichtet ASAN im Hintergrund jedoch seine Arbeit und prüft auf Fehler zur Laufzeit.

Im Allgemeinen findet ASAN Fehler, die der Programmierer bei der Speicherverwaltung macht, z. B. Buffer-Overflows (http://openbook.rheinwerk-verlag.de/linux_unix_programmierung/Kap18A-006.htm), Use-After-Frees (https://cwe.mitre.org/data/definitions/416.html) oder auch Memory-Leaks (http://openbook.rheinwerk-verlag.de/c_von_a_bis_z/027_c_sicheres_programmieren_002.htm).

Im Folgenden soll die Funktionalität und Verwendung von ASAN beispielhaft an einem Buffer-Overflow vorgestellt werden. Dafür wird das folgende (triviale) Programm test.c verwendet:

```
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv) {
        char buffer[32];
        strcpy(buffer, argv[1]);

}
```

Das Programm nimmt einen String über die Kommandozeile entgegen und kopiert ihn in ein 32-Byte großes Array. Da strcpy keine Längenüberprüfung durchführt gibt es ein simples Problem, wenn der übergebene String länger als 31 Zeichen ist: Der Buffer ist nicht groß genug. In C gibt es jedoch keinen Mechanismus der einen solchen Fall verhindert, weshalb strcpy den gesamten String auch über die 32 Byte des Arrays hinausschreibt und somit in Speicherbereiche schreibt, die nicht für normale Variablen-Daten reserviert wurden. Man sagt, dass der Buffer „überlauft“ (→ buffer overflow), da mehr Daten in ihn reingeschrieben werden als er halten kann und die überlaufenden Daten andere Speicherbereiche überschreiben. Durch einen solchen Fehler können speziell geformte Eingaben dafür sorgen,

dass das Programm beliebigen Code ausführt, der von der eigentlichen Programmierung abweicht.

Kompiliert man das Test-Programm wie folgt, kann man den Fehler in der Praxis sehen:

clang –g –o test test.c

Führt man das Programm mit einem Input der kürzer als 32 Bytes ist aus, passiert erstmal nichts:

```
~$ ./test AAAA
~$
```

Übergeben wir einen String der >= 32 Bytes ist, können wir beobachten, dass das Programm abstürzt (was hier noch vom Betriebssystem abgefangen wird):

```
~$ ./test AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
   *** stack smashing detected ***: <unknown> terminated
   Aborted (core dumped)
```

Dieser Absturz ist durch den zuvor erklärten Bug bedingt, da der Programmierer es versäumt hat sicherzustellen, dass der zu kopierende String nicht größer als das Ziel-Array ist.

Mit diesem Wissen könnte man erwarten, dass das Programm auch abstürzt, wenn man genau 33 Bytes als Eingabe gibt, da der Buffer dadurch bereits überlauft. In der Realität stellt man allerdings fest, dass dies nicht der Fall ist:

```
~$ ./test AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
~$
```

Obwohl durch unsere Eingabe der 32-Byte große Buffer überläuft, bleibt dies in diesem Fall unbemerkt und das Programm läuft einfach weiter. Der Bug existiert jedoch trotzdem, da er allerdings von der genauen Eingabe abhängt kann er leicht übersehen werden. Aus diesem Grund kommt der AddressSanitizer zum Einsatz.

Um das zu demonstrieren wird das Programm jetzt mit ASAN kompiliert:

clang –g –o test test.c –fsanitize=address

Führt man das Programm jetzt erneut mit der 33-Byte großen Eingabe aus, kann man sehen, dass der AddressSanitizer eingreift, uns den Fehler mit ausführlichen Informationen meldet und das Programm an dieser Stelle beendet:

```
~$ ./test AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
=================================================================
==102==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7fffebbcfa60 at pc 0x0000004aad4c
WRITE of size 34 at 0x7fffebbcfa60 thread T0
    #0 0x4aad4b  (/home/julian/test+0x4aad4b)
    #1 0x51220b  (/home/julian/test+0x51220b)
    #2 0x7f166e621b96  (/lib/x86_64-linux-gnu/libc.so.6+0x21b96)
    #3 0x419d19  (/home/julian/test+0x419d19)

Address 0x7fffebbcfa60 is located in stack of thread T0 at offset 64 in frame
    #0 0x5120ef  (/home/julian/test+0x5120ef)

  This frame has 1 object(s):
    [32, 64) 'buffer' (line 5) <== Memory access at offset 64 overflows this variable
```

Roter Kasten: ASAN sagt uns welche Art von Fehler abgefangen wurde, hier ein Stack-Buffer-Overflow.

Brauner Kasten: ASAN sagt uns auch welcher Speicherzugriff den Fehler ausgelöst hat. In diesem Fall war aus ein schreibender Zugriff, der 34 Bytes schreiben wollte (unsere 33 Byte lange Eingabe + Nullterminator).

Gelber Kasten: Da das Programm mit Debug-Symbolen kompiliert wurde (Flag –g), wird auch mitgeteilt welcher Speicherbereich von dem Fehler betroffen ist, hier unser Array „buffer“, deklariert in Zeile 5 des Quellcodes.

Diese ausführlichen Informationen können jetzt verwendet werden, um das Problem zu beheben. Da wir nun wissen, dass unser Buffer überlaufen kann, können wir das Kopieren des Eingabe-Strings in diesen Buffer absichern, indem wir maximal 31 Zeichen dorthin kopieren. Außerdem setzen wir das letzte Byte unsere Arrays als Null-Byte, um sicherzugehen, dass unser String nullterminiert ist:

```
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv) {
        char buffer[32];
        strncpy(buffer, argv[1], 31);
        buffer[31] = '\0';

}
```

Dieses Beispiel ist zwar nicht repräsentativ für alle Fehler-Typen, die der AddressSanitizer finden kann, jedoch liefert er in den meisten Fällen genug Informationen, um Art und Ursache des Fehlers herausfinden und beheben zu können. Da ein Bug nicht bei jeder Eingabe ein verändertes Verhalten des Programmes verursacht und sich somit bemerkbar macht, sollte ASAN während der Entwicklung verwendet werden. Durch die sehr strengen Kontrollen die der AddressSanitizer zur Laufzeit durchführt, können somit auch gut versteckte Fehler gefunden werden.