

Versuch A: Buffer Overflow

Lara Quitte, Tino Jeromin

Hintergrundwissen

Ein Programm in C ist in folgende Speicherbereiche aufgeteilt:

- Stack
- Heap
- Uninitialized data
- Initialized data
- Text

Diese Bereiche sollen im Folgenden kurz näher betrachtet werden. Ausserdem soll ein Buffer-Overflow sowie Stack- und Heap-Overflow erläutert werden.

Heap

Der Heap wird in C genutzt, um dynamisch Speicher für Variablen, Arrays, etc zu reservieren. Es wird also während der Laufzeit des Programms Speicher reserviert und wieder freigegeben. Dies geschieht mittels 'malloc', 'realloc' und 'calloc'. Die so angelegten Variablen und ihr Speicherplatz existieren bis sie vom Programm mittels 'free' wieder freigegeben werden. Werden die angelegten Variablen nicht wieder freigegeben, kann es zu Memory Leaks kommen, d. h. das Programm reserviert immer mehr Speicher, gibt ihn aber nicht wieder frei, bis kein Speicherplatz mehr verfügbar ist.

Stack

Auf dem Stack werden die Funktionen und die dazugehörigen Informationen gespeichert. Ruft also ein Programm eine Funktion auf, wird die Rücksprungadresse, die übergebenen Parameter und alle lokalen Variablen auf den Stack gelegt. Diese Informationen werden zusammen Stack Frame genannt, was jede Funktion besitzt. Ist eine Funktion beendet, wird der Stack Frame wieder vom Stack genommen und das Programm springt zu der Funktion, die diese Funktion aufgerufen hat. Das Legen einer Variable auf den Stack wird als 'push' bezeichnet und das entfernen vom Stack als 'pop'. Das Betriebssystem speichert typischerweise mittels eines Zeigers den Anfang des Stacks und das aktuell oberste Element des Stacks. Der Stack ist je nach Betriebssystem und CPU unterschiedlich implementiert.

Uninitialized data

Hier werden alle globalen und statischen Variablen gespeichert, welche nicht initialisiert wurden, oder zu 0 initialisiert wurden.

Initialized data

Hier werden alle globalen und statischen Variablen gespeichert, welche vom Programmierer initialisiert wurden. Zusätzlich gibt es einen Bereich für read-write Zugriff und einen für read-only Zugriff.

Text Segment

In diesem Segment wird der Code des Programms gespeichert, also die ausführbaren Befehle. Üblicherweise ist dieses Segment read-only, damit es nicht im Falle von stack- oder heap-overflows überschrieben wird.

Buffer Overflow

Ein Buffer in C ist typischerweise ein Array mit einer festen Größe. Wird nun eine Datenmenge, die größer ist als der Buffer, in den Buffer gespeichert, ohne dass der Speicherplatz überprüft wird, dann kommt es zu einem Buffer-Overflow. Dabei wird der Speicherplatz hinter dem Buffer auch noch beschrieben, welcher gar nicht mehr für den Buffer reserviert ist und andere Informationen enthalten kann, die überschrieben werden.

Stack Overflow

Bei einem Stack-Overflow werden Daten an eine Stelle geschrieben, die nicht dafür vorgesehen sind. Zum Beispiel wenn in einen Buffer mehr Daten geschrieben werden sollen, als dafür reserviert ist. Ein Angriff, welcher sich einen Stack-Overflow zu Nutze macht, basiert beispielsweise auf einem Buffer-Overflow. Dabei werden Daten in einen Buffer geschrieben, der auf dem Stack liegt.

Da bei einem Buffer-Overflow über den Buffer hinaus auf den Stack geschrieben wird, kann dadurch auch die Rücksprungadresse verändert werden. Verlangt ein Programm ein Array(String) als Parameter, wie zum Beispiel ein Programm, was über eine Shell ausgeführt wird und kopiert dieses Array ohne die Länge zu überprüfen, dann wird das Eingabe-Array auf den Stack kopiert und überschreibt je nach Länge auch die Rücksprungadresse. Die Funktion 'strcpy' verwendet zum Beispiel kein Bounds-Checking. Auf diese Weise kann die Rücksprungadresse so verändert werden, dass die in den Buffer zeigt, der ja das Eingabe-Array enthält. Wenn dieses Eingabe-Array nun aus ausführbaren Instruktionen besteht, werden diese statt der eigentlichen ausgeführt.

So kann beispielsweise eine Shell aufgerufen werden, auf welcher dann beliebige weitere Programme ausgeführt werden können. Wurde das ausgenutzte Programm als root installiert, würde die aufgerufene Shell sogar root-Rechte haben.

Heap Overflow

Ein Heap-Overflow tritt auf, wenn mehr Daten in den Heap geschrieben werden sollen, als Speicherplatz für den Heap verfügbar ist.

Teil 1: Non-Control Data Attack (Buffer Overflow)

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

1. Öffne Ghidra
2. Schliesse Help Window
3. Neues Project erstellen
4. Import 'ncd' executable
5. Öffne ncd im CodeBrowser
6. Symbol Tree → Exports → main
7. Window → Decompile: main

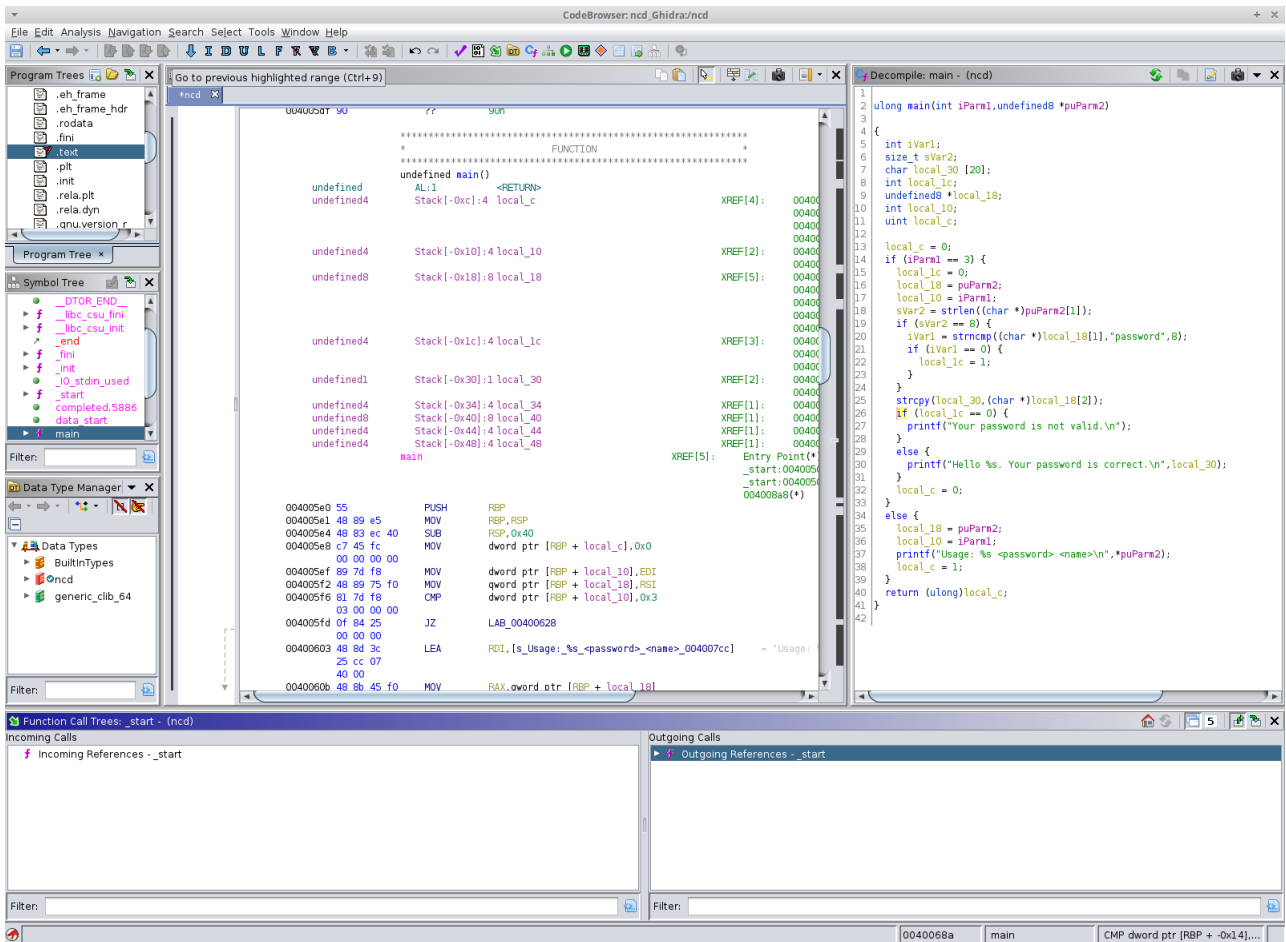


Figure 1: ncd decompiled in Ghidra

Nach Schritt 7 erhält man folgenden Code:

```

1  ulong main(int iParm1, undefined8 *puParm2)
2
3  {
4      int iVar1;
5      size_t sVar2;
6      char local_30 [20];
7      int local_1c;
8      undefined8 *local_18;
9      int local_10;
10     uint local_c;
11
12     local_c = 0;
13     if (iParm1 == 3) {
14         local_1c = 0;
15         local_18 = puParm2;
16         local_10 = iParm1;
17         sVar2 = strlen((char *)puParm2[1]);
18         if (sVar2 == 8) {
19             iVar1 = strcmp((char *)local_18[1], "password");
20             if (iVar1 == 0) {
21                 local_1c = 1;
22             }
23         }
24         strcpy(local_30, (char *)local_18[2]);
25         if (local_1c == 0) {
26             printf("Your password is not valid.\n");

```

```

27     }
28     else {
29         printf("Hello %s. Your password is correct.\n", local_30);
30     }
31     local_c = 0;
32 }
33 else {
34     local_18 = puParm2;
35     local_10 = iParm1;
36     printf("Usage: %s <password> <name>\n", *puParm2);
37     local_c = 1;
38 }
39 return (ulong)local_c;
40 }

```

Wählt man als Parameter irgendeinen Namen und als Passwort eine Zeichenfolge, die länger als 20 Zeichen ist, hält das Programm die Kombination für korrekt.

Mithilfe des von Ghidra erzeugten C-Codes lässt sich die Funktionsweise des Programms gut nachvollziehen. Zu Beginn der 'main' Funktion werden verschiedene lokale Variablen deklariert, für welche der benötigte Speicherplatz auf dem Stack hintereinander reserviert wird. Dabei wird der Buffer 'local_30' mit einer Größe von 20 vor dem Integer 'local_1c' auf den Stack gelegt. Die kritische Stelle ist in Zeile 25, wenn mittels 'strcpy' unser eingegebenes Passwort in den Buffer 'local_30' kopiert werden soll. Da 'strcpy' jedoch kein Bounds-Checking betreibt, überschreiben Passworteingaben, die länger als 20 Zeichen sind, die Variablen, die nach dem Buffer auf dem Stack liegen. In diesem Fall als erstes 'local_1c'. 'local_1c' wird aber in Zeile 26 in einer if-Abfrage dazu genutzt wird, die Korrektheit des Passwortes festzustellen. Ist 'local_1c' gleich '0', ist das Passwort falsch. Sonst ist das Passwort richtig. Da durch den Buffer-Overflow 'local_1c' mit einem Zeichen überschrieben wurde, welches nicht '0' ist, verhält sich das Programm, als ob das Passwort richtig wäre.

Eine Möglichkeit, den Angriff zu verhindern, ist, 'strncpy' statt 'strcpy' zu verwenden. Alternativ könnte vor dem Aufruf von 'strcpy' die Länge des Strings überprüft werden. Ausserdem könnten skalare Variablen vor Buffern deklariert werden, sodass die skalaren Variablen vor den Buffern auf dem Stack liegen und so bei einem Buffer-Overflow keine skalaren Variablen überschrieben werden können. Eine weitere drastische Maßnahme wäre, eine andere Programmiersprache wie Java oder eine interpretierte Sprache zu verwenden, da bei diesen Buffer-Overflows nicht möglich sind.

Eine Möglichkeit, Buffer-Overflows zu entdecken, ist jede Eingabemöglichkeit mit sehr langen Eingaben zu testen, um einen Buffer-Overflow zu provozieren. Eine einfache Möglichkeit, jeden möglichen Buffer-Overflow zu entdecken, gibt es jedoch nicht. Um Programme auf diese Sicherheitslücke zu untersuchen, muss sich jede Verwendung von Buffern genau angeschaut werden.

Um zu verhindern, dass ein Angreifer auf den gesamten Arbeitsspeicher zugreifen kann, benutzt Linux zum Beispiel Stack Canaries. Dies sind bestimmte Werte, die auf dem Stack zwischen der Rücksprungadresse und den lokalen Variablen gelegt werden. Diese Werte werden regelmäßig, zum Beispiel vor dem Sprung zur Rücksprungadresse, überprüft. Ist der Wert verändert worden, wird das Programm abgebrochen. SO wird verhindert, dass die Rücksprungadresse verändert wird. Zudem verwendet Linux Paging und Virtual Memory. Der Arbeitsspeicher wird also in Blöcke aufgeteilt, welche eine virtuelle Adresse haben. Versucht ein Programm auf eine Adresse ausserhalb des eigenen Blocks zuzugreifen, entsteht ein Page Fault, sodass das Programm unterbrochen wird.