michael.engel [☺] hs-coburg.de

Praktikum zur Vorlesung
Malware-Analyse und Reverse Engineering
Sommersemester 2017

Praktikumsaufgabe 6

Besprechung am Donnerstag, 11. Mai 2017

Hinweise: Praktikumsaufgaben müssen in der vorgegebenen Zeit in den eingeteilten Gruppen bearbeitet werden. Die Lösung der Aufgaben wird bewertet, jedes Gruppenmitglied soll Fragen zu den einzelnen Aufgaben beantworten können. Bei Unklarheiten und Rückfragen wird selbstverständlich eine Hilfestellung gegeben.

6.1 Kontrollflussverfolgung

Eine Methode, die Gefahren von Return Oriented Programming und der Veränderung von Rücksprungadressen auf dem Stack zu mindern, ist die Einführung von Kontrollflussverfolgung für Funktionen.

1. Schreiben Sie ein kleines Testprogramm, das Fibonacci-Zahlen *rekursiv* berechnet. Das Programm soll eine Eingabe von der Tastatur einlesen, die den Parameter `n` für die Berechnung von `fib(n)` einliest. Gleichzeitig soll Ihr Programm, um Rechenzeit zu sparen, nur Eingaben von $n <= 5$ zulassen.

2. Entwickeln Sie eine Funktion, die es ermöglicht, die Funktionsaufrufhierarchie eines (beliebigen) C-Programms zur Laufzeit auszugeben. Verwenden Sie dazu die Compiler-Option `-finstrument-functions`.

   Diese gcc-Option ruft bei Betreten (enter) und vor dem Verlassen (exit) von Funktionen jeweils eine von Ihnen zu definierende Funktion auf:

   ```
   void __cyg_profile_func_enter (void *this_fn, void *call_site);
   void __cyg_profile_func_exit  (void *this_fn, void *call_site);
   ```

   Die übergebenen Parameter sind die Adresse der aktuellen (instrumentierten) Funktion sowie die Adresse, von der aus die aktuelle Funktion aufgerufen wurde.

   Geben Sie in der Ausgabe jeweils Adresse (optional: Namen) der aufrufenden und aufgerufenen Funktion an.

   Welches Problem fällt Ihnen auf?

3. Entwickeln Sie darauf basierend eine Funktion, die den Kontrollfluss bei Funktionsaufrufen schützt, indem Aufrufe von “falschen” Funktionen/Adressen aus erkannt werden und zu einem Programmabbruch führen. Sie können die zulässige Funktionsaufruf- Hierarchie dazu statisch (z.B. als array/struct) definieren.

4. Versuchen Sie, diesen Schutz zu umgehen, damit durch Berechnung riesiger Fibonacci-Zahlen unzulässig wertvolle Rechenzeit verschwendet werden kann!

Denken Sie wie immer daran, Ihren Code mit `-fno-stack-protector` zu übersetzen!