Buffer Overflow

Schwachstellen und wie man sie Schließt

Jakob Stühn, John Meyerhoff, Sam Taheri

H-BRS

Inhaltsverzeichnis

1 Abstract

Buffer Overflows sind trotz ihres hohen Alters noch immer eine der, wenn nicht sogar die relevanteste, Schwachstelle in Computerprogrammen. Aus diesem Grund ist es unabdinglich für jeden der sich im Feld der Software-Entwicklung oder IT-Sicherheit bewegt, ein Grundlegendes Verständnis für Buffer Overflows aufzubauen. Ein Überblick über große und aktuelle Angriffe zeigt, wie verheerend die Auswirkungen einer solchen Schwachstelle, in den falschen Händen, sein kann.
Der vorliegende Projektbericht beschäftigt sich deshalb, mit einer theoretisch, technischen Einführung, sowie praktischen Analyse von Buffer Overflow Schwachstellen. Ein Buffer Overflow beschreibt

hierbei im Kern, das “Überlaufen” eines Speicherbereiches durch unvorhergesehene Eingaben, wodurch Schadcode in einen laufenden Prozess injiziert und ausgeführt werden kann. Um eine Grundlage für praktische Tests zu schaffen, wurde zunächst die Struktur und der Ablauf eines Programms im Speicher analysiert und theoretische Angriffsmöglichkeiten konstruiert. Anschließend wurde ein fiktives Angriffsszenario und ein verwundbares C-Programm entwickelt. Hierbei zeigte sich schnell, das bereits die Nutzung von vermeintlich harmlosen Funktionen, wie fprint() oder gets(), schwerwiegende Auswirkungen auf die Sicherheit einer Applikation haben kann. Um die zuvor konstruierten Angriffstechniken, real zu erproben und die Sicht eines Angreifers möglichst realistisch zu analysieren, wurde das verwundbare Programm anschließend im GNU-Debugger durchleuchtet. Dabei ließ sich klar erkennen, dass der Angreifer, von einer tatsächlichen Kopie des anzugreifenden Programms, wenn nicht sogar des Source-Codes, profitiert. Mit dem aus dem Debugger erlangten Wissen, wurde nun ein Exploit gebaut und in einer simulierten Server-Umgebung ausgeführt. Mit der Zuhilfenahme eines injizierten Assembler Programms, konnte auf dem Zielsystem erfolgreich eine privilegierte Shell geöffnet werden. Abschließend wurde sich mit den wichtigsten Abwehrmechanismen von Compiler und Betriebssystem auseinandergesetzt, sowie “cutting-edge” Präventions-Mechanismen, wie KI gestützte Codeanalyse, untersucht. Hier zeigte sich klar, dass einem Angreifer die Arbeit zwar erschwert werden kann, Buffer Overflows jedoch nie vollständig verhindert werden können.
Durch diese auf praktische Beispiele fokussierte Herangehensweise, sollte der Leser dieses Projektberichts die Thematik der Buffer Overflows besser verstehen und einen direkten Nutzen für seine Arbeit ziehen können.

2 Grundaufbau

- Eingabemöglichkeit
- Speichern der Eingabe
- Ablegen von Anweisungen durch übergroße Eingabe
- Ausführen der Anweisungen → Remote Code Execution

3 Geschichte

3.1 Bekannte Buffer-overflows

3.2 Aktuelle Beispiele

4 Grundlegende Theorie

4.1 Aufbau des Stacks

4.2 Angriffsvorgang

4.3 Verhalten bei Overflow

5 Shellcode

5.1 Code

5.2 Erläuterung

6 Anwendungsfallbeispiel

6.1 Code

6.2 Setup des Servers

6.3 Böswilliger Client

6.4 Erläuterung des Vorgangs

7 Gegenmaßnahmen

Wie in Abschnitt 6.4 bereits aufgeführt, gehören zu einem Buffer-Overflow-Angriff mehrere kombinierte Teile. Wenn man nun verhindern möchte, dass ein Programm über diesen Angriff gestürzt werden kann, so hat man mehrere Möglichkeiten diese Teile aufzuhalten oder die Kombination zu blockieren. Es folgen mehrere Möglichkeiten, grob nach Aufwand (für den Entwickler) sortiert.

7.1 Übersicht der Maßnahmen

Low Level:

- Hardware-basierte Lösungen
- Betriebssystembasierte Ansätze

Laufzeitlösungen:

- C Range Error Detector und Out Of Bounds Object
- Safe Pointer Instrumentalisierung
- Manuelles Buffer-Overflow Blocken (Input-Bereinigung)

Vorbereitete Lösungen:

- Statische Code-Analyse
- Stack-Schutz mit “Canary” (Zufallszahl)

7.2 Canaries

Die Bezeichnung Canary (Engl. Kanarienvogel) stammt aus der Verwendung der Kanarienvögel als Indikator für Gas in Mienen. Die Canaries im Code werden als Stack-Schutz verwendet. Das bedeutet, dass beispielsweise Zufallszahlen im Programm auf dem Stack sind und bei einem Buffer-Overflow-Angriff überschrieben werden. Ein Tool wie StackGuard kann dann anhand der Änderung einen Fehler feststellen und die Ausführung des Programms abbrechen.

7.3 Out Of Bounds Object

Ein Out Of Bounds Object ist eine Vereinfachte Lösung um Referenzen ungefährlich zu machen. Es wird verhindert, dass auf Speicher außerhalb des Programms zugegriffen wird, indem Jede Adresse welche nicht im spezifizierten Bereich liegt auf ein bestimmtes Objekt, das sogenannte Öut Of Bounds Object"verweist. Dadurch kann innerhalb des laufenden Programms erkannt werden, das etwas falsch gelaufen ist. Diese Methode ist nicht gängig, da sie technisch gesehen umgangen werden kann, solange der Angreifer weiß, welcher Speicherbereich für das Programm vorgesehen ist.

7.4 Code-Beispiel

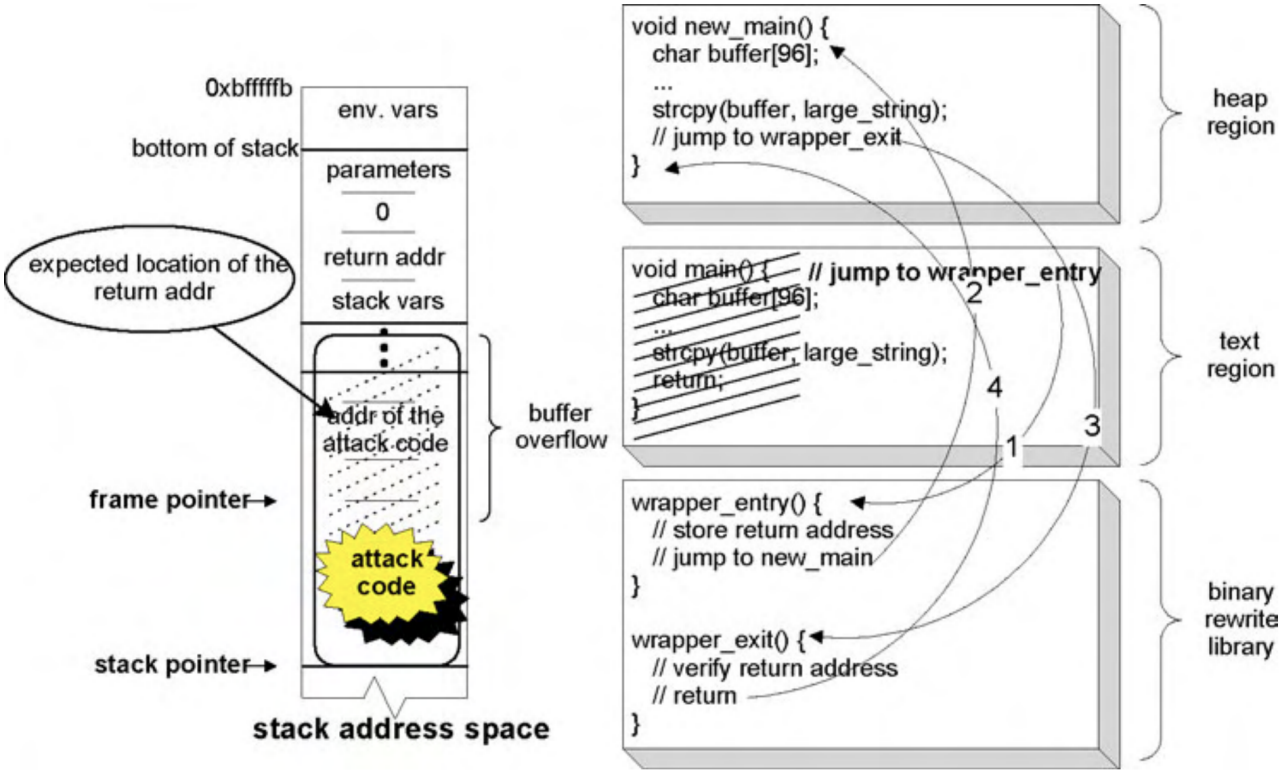

Figure 9: Libverify funtion call and stack layout

7.5 Testen

- Fuzzy Tests
- Spezifische Payloads

8 Quellen

- https://www.nds.ruhr-uni-bochum.de/media/nds/attachments/files/2010/11/Survey.on.Buffer.Overflo