Studiengang Informatik **Fachhochschule Nordwestschweiz**
apsi 2021 Arno Wagner, Michael Schläpfer

Lösungsblatt 2 4. Oktober 2021

Aufgabe 2-1: Buffer Overflow

i) Simple Buffer Overflow

(a) Ein Segmentation Fault tritt auf, wenn ein Programm versucht auf einen eingeschränkten Speicherbereich zuzugreifen. Im Beispiel wird die Rücksprungadresse überschrieben, in der Folge wird nach Beendigung von `test()` versucht auf einen ungültigen Speicherbereich zuzugreifen.

(b) Nach dem Setzen eines Breakpoints auf Zeile 7 können die Werte `s1` und `s2` jeweils mittels der GDB Anweisung `display` ausgelesen werden. Die Rücksprungadresse kann jeweils mittels `info frame` ausgelesen werden. Dabei ist die Rücksprungadresse in der Variable `saved rip` ersichtlich.

(c) Der Segmentation Fault tritt in Zeile 9 auf, wenn die Rücksprungadresse gelesen und versucht wird, zur weiteren Programmausführung auf den entsprechenden Speicherbereich zuzugreifen.

(d) Mit `info frame` erhält man eine Übersicht über den Stackframe. Die Variable `saved rip` gibt dabei Aufschluss über die gespeicherte Rücksprungadresse. Nach Ausführung von Zeile 7 ist die Rücksprungadresse überschrieben.

ii) Buffer Overflow Exploit Das Programm erlaubt eine Kommandozeilen-Eingabe und schreibt diese Eingabe dann in einen 20 Byte grossen Buffer. Anschliessend wird der Buffer wiederum auf der Kommandozeile ausgegeben. Die `secretFunction()` wird dabei nie aufgerufen.

```
...
080488b5 <secretFunction>:
...

080488f1 <echo>:
 ...
 8048918:       8d 45 e4                lea    -0x1c(%ebp),%eax
 ...
```

Die Adresse in Hexadezimalschreibweise der `secretFunction()` beträgt in obigem Beispiel `0x080488b5`. Daraus lässt sich folgende Adresse ableiten, welche als Rücksprungadresse eingefügt werden soll: `\xb5\x88\x04\x08`.

Die `lea` Instruktion (Load Effective Address) gibt die Position des `buffer` relativ zum `ebp` (Base Point des aktuellen Stacks) an. In unserem Beispiel `-0x1c`, als 28 Dezimal. Wie im Tutorial beschrieben befindet sich unter dem `ebp`, der Grösse 4 Bytes die Rücksprungadresse. Somit ergeben sich, wie im Tutorial 32 Bytes, welche gefüllt werden müssen, bevor die Rücksprungadresse überschrieben werden kann. Mit dem nachfolgenden Aufruf wird nun die Rücksprungadresse erfolgreich überschrieben:

```
$ perl -e 'print "a"x32 . "\xb5\x88\x04\x08"' | ./vuln
Enter some text:
You entered: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa??
Congratulations!
You have entered in the secret function!
```

iii) Return Oriented Programming

Return-oriented Programming erlaubt es, durch das Überschreiben von Paramtern und Rücksprungadressen einen beliebigen Programmablauf zu erstellen, womit kein eigener Schadcode eingefügt und ausgeführt werden muss.

Aufgabe 2-2: Algorithmic Complexity Attacks

i) Hashtables

(a) Im optimalen Fall wächst die Zeitkomplexität des Zugriffs auf die Werte in einer Hashtable nicht mit der Anzahl von Einträgen, sondern bleibt konstant, also $O(1)$. Wenn die Hashwerte sämtlicher Einträge zum selben Bucket führen, muss für alle Werte die Liste des Buckets durchsucht werden, womit die Zeitkomplexität linear zur Anzahl Einträge wächst, also $o(n)$.

(b) a) Performance, b) Gleichverteilung des Outputs um Kollisionen zu vermeiden.

(c) Eine einfache und oft gewählte Möglichkeit um Kollisionen aufzulösen ist die *Verkettung*, dabei werden die kollidierenden Einträge einfach in eine Liste eingetragen. Eine weitere Möglichkeit ist aber anhand einer Strategie nach einer freien Position weiterzusuchen. Hier gibt es verschiedene Verfahren, wie *lineares oder quadratisches Sondieren* oder *doppeltes Hashen*.

(d) Kryptographische Hashfunktionen haben eine wesentlich schlechtere Performance, weshalb sie für den Anwendungsfall von Hashtables in der Regel nicht geeignet sind.

ii) ReDoS Attacks

(a) Regex werden fast überall verwendet, wo es darum geht Eingaben zu verarbeiten oder zu prüfen. In den meisten Webanwendungen und auf jedem Layer werden Regex eingesetzt.

(b) Eine schlechte Regex beinhaltet Gruppierung mit Wiederholung.

(c) Kosten werden in Cloud-Computing Umgebungen unter anderem verbunden mit der effektiven Rechenleistung erhoben. Ein Konkurrent könnte so beispielsweise die Kosten hochalten, indem er entsprechende Anfragen automatisiert stellt.

iii) Email Address Validation

(a) Nein, die vorgeschlagene Regex ist nicht komplett. Die Mailadresse michael.schlaepfer@fort-it.ch beispielsweise wird nicht gematched.

(b) Verwendet man beispielsweise `regex101.com`, so sieht man schnell, dass für ungültige Maildressen von der Form `xxxx-@fhnw.ch` mit jedem weiteren `x` eine Verdoppelung der Anzahl Schritte zur Prüfung durchgeführt werden.

(c) Sie zeigen Ihrem Kollegen zuerst die Anzahl Schritte für `aaaa-@fhnw.ch` (87 Schritte) und anschliessend für `aaaaaaaaaaaaaaaaa-@fhnw.ch` (688'131 Schritte).