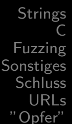
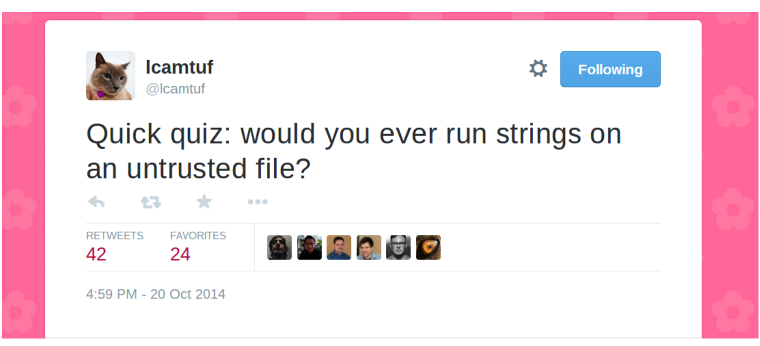


Fuzzing für Anfänger

Hanno Böck

Strings

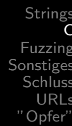


CC ZERO

Spaß mit binutils

- Was viele nicht wissen: strings parst executables mit libbfd (binutils) und gibt deren Struktur aus
- 87 Parser-Bugs in libbfd (objdump, nm, strings) - bisher kein Ende in Sicht
- CVE-2014-8484, CVE-2014-8485, CVE-2014-8501, CVE-2014-8502, CVE-2014-8503, CVE-2014-8504
- https://sourceware.org/bugzilla/show_bug.cgi?id=17510
  https://sourceware.org/bugzilla/show_bug.cgi?id=17512
  https://sourceware.org/bugzilla/show_bug.cgi?id=17533

CC ZERO



C Memory Bugs

- Buffer Overflow, Stack Overflow, Heap Overflow, Use-after-Free, Out-of-bounds, Memory Corruption, Off-by-1, ...
- Zusammengefasst: Software liest oder schreibt an der falschen Stelle
- Viele Sicherheitslücken sind Fehler im C-Speichermanagement

Beispiel OOB-Error in C

```
int main() {
  int i[3] = {3, 1, 2};
  printf("%i\n", i[3]);
}
```

C abschaffen?

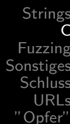
- Lösung: Wir schreiben alles (also wirklich *alles*) neu in einer Programmiersprache mit besserer Speicherverwaltung (Lisp, OCaml, Go, Rust, Python, Erlang, Dylan, Haskell, F#, Ada, ...)

CC ZERO

C Realitätscheck

- Alle relevanten Betriebssysteme und alle Browser sind in C/C++ geschrieben
- Und noch viel mehr
- Projekte die wirklich versuchen relevante Core-Projekte in sicheren Sprachen neu zu schreiben sind rar (Interessant, aber experimentell: miTLS)

CC ZERO



C sicher machen

- Mitigation: Stack Protector, Stack Canary, ASLR, Fortify Source, Pax, NX pages, Address Sanitizer, Softbound+CETS, ...
- Prinzipiell gut, aber alle Ansätze sind entweder unvollständig oder nicht praktisch einsetzbar

Fuzzing

- Die Idee: Wir erstellen massenhaft fehlerhafte Eingabedaten und schauen was passiert (Crash, Hang)
- Crash bei fehlerhaften Daten ist oft Hinweis auf eine Sicherheitslücke
- Man kann dabei viele fortgeschrittene Methoden verwenden, aber die lassen wir beiseite, Denn die allereinfachsten genügen bereits

CC ZERO

Geeignete Ziele

- Gut Fuzzen lassen sich Tools die komplexe Binärformate parsen (Bilder, Packer, Executables, ...)
- Je mehr exotische Formate desto besser

CC ZERO

Die simpelste Idee

- /dev/urandom ?
- Manchmal reicht das bereits
- (zugehöriger Bug ist an die Entwickler gemeldet, aber noch nicht öffentlich)

CC ZERO

zzuf

- Wir besorgen ein paar Dateien (input.*) und erstellen uns defekte Samples:

```
for f in input.*; do for i in {1000..3000};
    do zzuf -s $i < $f > $i-$f; done; done
```

- Wir testen unsere Samples:

```
export LC_ALL=C;  for f in *; do timeout 2
   [path_to_tool] $f;echo $f;done &>
   fuzzlog
```

- grep -B1 "Segmentation fault" fuzzlog
- (man kann zzuf direkt Programm testen lassen, macht aber mit Address Sanitizer Probleme)

CC ZERO

Analyse: Valgrind

- `valgrind [path_to_tool] [crash_sample]`
- valgrind ist langsam, aber gründlich.
- Ausgabe relativ gut verständlich, besser bei Programmen mit Debugging-Infos (CFLAGS="-g").

CC0 ZERO

Address Sanitizer

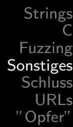
- Bisherige Methode bereits oft erfolgreich, aber es geht noch viel besser
- Was passiert eigentlich bei unserem Beispielprogramm? Es liest einfach aus dem falschen Speicherbereich - kein Crash!
- Address Sanitizer (asan) führt zusätzliche Bounds-Checks bei gcc/llvm ein - Programm etwa doppelt so langsam

CC ZERO

Address Sanitizer

- `./configure --disable-shared CFLAGS="-fsanitize=address -g" LDFLAGS="-fsanitize=address -g"; make`
- Skript zum Output decodieren (scheinbar bei gcc 4.9 nicht mehr nötig): `http://llvm.org/klaus/compiler-rt/raw/release_35/lib/asan/scripts/asan_symbolize.py`
- Beachten: grep nach "AddressSanitizer" statt "Segmentation fault"
- AddressSanitizer verträgt sich nicht immer mit anderen Tools, bspw. valgrind, zzuf (Direktaufruf)

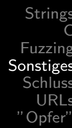
CC ZERO



Bewertung von Bugs

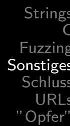
- Faustregel 1: Schreibzugriffe sind problematischer als Lesezugriffe (Es gibt Ausnahmen: Heartbleed)
- Faustregel 2: Es ist oft einfacher Bugs zu finden und zu fixen als herauszufinden ob sie exploitbar sind
- "All bugs should be fixed" - Diskussionen darüber wie kritisch Bugs sind im Zweifel sparen

cc 0 ZERO



american fuzzy lop

- Versucht hohe Abdeckung von Codepfaden zu erreichen
- Recompile mit eigenem Compiler-Wrapper nötig, scheitert manchmal und erfordert manuelle Anpassungen, nicht ganz ausgereift
- Nice: Gruppiert Crash-Samples gleich nach vermutlich identischen Bugs



Was noch?

- Checksummen können Fuzzing verhindern - rauspatchen oder neu berechnen (umständlich)
- Natürlich kann man auch andere Eingaben (Netzwerk, USB) fuzzen, aber komplizierter
- Gezielt bestimmte Felder in Dateiformaten fuzzen, bspw. mit Extremwerten (0, 0xffff...), erfordert angepassten Fuzzer pro Dateiformat, umständlich
- Es ist oft eine gute Idee die Crash-Samples von einem Tool mit einem anderen Tool zu testen
- 32 vs. 64 Bit - 32 Bit findet oft mehr Probleme

Ziel

- Ziel: In einem Standard-Linux-System sollten sich mit trivialem Fuzzing keine Speicherzugriffsfehler mehr finden lassen.
- Mitstreiter gesucht!

CC ZERO

URLs

- http://libcaca.zoy.org/wiki/zzuf/
- https://code.google.com/p/american-fuzzy-lop/
- https://code.google.com/p/address-sanitizer/

"Opfer"

- ImageMagick CVE-2014-8354, CVE-2014-8355, CVE-2014-8562
- gdk-pixbuf / claws-mail
  `https://bugzilla.gnome.org/show_bug.cgi?id=739785`
- GraphicsMagick `http://sourceforge.net/p/graphicsmagick/code/ci/37ab9576dbdfeecd8bbc0a312a49b362846016c1/`
- GIMP `https://bugzilla.gnome.org/show_bug.cgi?id=739133`
  `https://bugzilla.gnome.org/show_bug.cgi?id=739134`
- elfutils `https://lists.fedorahosted.org/pipermail/elfutils-devel/2014-October/004215.html`
- More (or less) to come