Firewalls mit perl

Steffen Ullrich
GeNUA mbH

Wer ist GeNUA?

- Hersteller von Security Appliances:
  - Hochsicherheitsfirewall GeNUGate
  - GeNUBox, GeNULink, GeNUScreen, GeNUCrypt...
- Sicherheitsmanagment und Systemwartung
- 15 Jahre alt, ca. 70 Mitarbeiter
- Perl als hauptsächliche Programmiersprache

Rolle von perl auf dem Firewall

- Kernel (OpenBSD) in C
- GUI, Auswertungen... in perl
- Application Level Gateways (Proxies) in perl:
  - WWW, POP, NNTP, SMTP, X.400, SIP, SNMP, FTP, Telnet
  - TCP, UDP, IP, ICMP (kein Forwarding)

Aufgaben der ALGs

- Schutz des inneren Netzes (Browser, Server,..):
  - vor gezielt fehlerhaftem Protokoll
  - vor gefährlichen Daten (Viren, Trojaner,...)
  - vor ambivalent interpretierbaren Daten
  - vor unerlaubten Zugriffen von innen
  - (vor unkontrollierter Weitergabe von Informationen)
- Selbstschutz des Firewalls:
  - mit stark eingeschränkten Rechten arbeiten
  - chroot, d.h. ALG in Käfig eingeschlossen

Herausforderungen

- unerwarteter Input
- chrooted
- hohe Zuverlässigkeit (Memory leaks..., da Programme ewig laufen)
- Lizenz vereinbar mit Lizenz des Firewalls

Lizenzen

- GeNUGate ist kommerzielles Produkt, welches auf einer Menge Open Source aufbaut (OpenBSD, perl+Module, squid, sendmail...).
- Lizenzen der Module meist wie bei perl
- bei einigen Modulen jedoch nicht explizit angegeben, was Nutzung evtl. ausschließt

Qualität des CORE

- perl CORE i.A. sehr stabil
- bisher keine Probleme bei Updates innerhalb perl5.8.x
- Reaktion auf Fehlerreports läßt zu wünschen übrig
    - Zwischen Bugreport und Fix vergehen Monate
    - auf einige Bugs (FD leaks) keinerlei Reaktion
    - teilweise nur in blead perl gefixt
    - Segmentation Faults werden nicht als kritisch betrachtet
- Leider selber nicht genügend Kenntnisse der perl-Interna.

Qualität der Module

- Verwenden i.A. nur Module, die im CORE oder aktuell gepflegt
- Die meisten Module, die wir verwenden haben eine sehr gute Qualität (Zuverlässigkeit, Benutzbarkeit, Dokumentation).
- Allerdings gibt es auch im perl CORE Module, die nur schlecht gepflegt sind
- Senden Bugreports und Patches an Maintainer, müssen aber teilweise Patches lokal weiterpflegen (bzw innerhalb von OpenBSD), da Maintainer nicht zeitnah reagieren.
- Übernehmen wenn sinnvoll auch die Maintainership (IO::Socket::SSL)

Perl in chroot Umgebungen

- kein Zugriff auf Dateien außerhalb der neuen Root
- daher kein dynamisches Nachladen

Dynamisches require vs. chroot

- ```
  use Carp 'croak';
  chdir( '/cage' );
  chroot( '/cage' ) || die $!;
  croak 'bad';
  ```

- `Can't locate Carp/Heavy.pm in @INC`
- auch Net::DNS, MIME::Decode,...
- Ausweg: alles vorher laden (man muss allerdings wissen was man braucht)
  ```

AutoLoader vs. chroot

- ```
  use POSIX 'exit';
  require Carp::Heavy; # see last example
  chdir( '/tmp' );
  chroot( '/tmp' ) || die $!;
  exit(0);
  ```

- `Can't locate auto/POSIX/exit.al in @INC`
- auch Storable, Net::SSLeay, ...
- Ausweg: vorher aufrufen wenn möglich, z.B.
  `eval{Storable::freeze()}`
  ```

Unicode vs. chroot I

- ```
  chdir( '/tmp' );
  chroot( '/tmp' ) || die $!;
  print shift(@ARGV) =~ m/(\p{IsPrint}+)/;
  ```

- `Can't locate utf8.pm in @INC`
- Ausweg: vorher explizit laden
  ```

Unicode vs. chroot II

- ```
  # html decode: &#48 -> '0'
  chdir( '/tmp' );
  chroot( '/tmp' ) || die $!;
  $_ = shift;
  s{  &\#(\d+) }{ chr($1) }xeg;
  print m{(\w+)};
  ```

- `test.pl 'bla&#48' -> 'bla0'`
- `test.pl 'bla&#480' -> 'Can't locate utf8.pm in @INC'`
- Problem tritt nur mit dem richtigen Input auf
- Ausweg: vorher explizit laden, use bytes
  ```

Reaktion in CPAN Modulen

- Module erwarten im Allgemeinen korrekten Input:
  - Reaktion bei fehlerhaftem oder ambivalentem Input u.U. problematisch
  - Superlange Zeilen als Input für MIME::Parser führen zu riesenhaftem Speicherverbrauch und schließlich zum Crash.
- Deshalb nur eingeschränkter Einsatz möglich:
  - Net::DNS spricht nur mit lokalem Nameserver
  - MIME::Parser in chroot und nur an Stellen, wo ein Crash den Rest des Systems nicht gefährdet
  - Benutzen i.A. eigenen Code, insb. weil es oft auch nötig ist Ambivalenzen vor dem Weiterleiten zu entfernen (z.B. HTML)

typische Fehler

- Einlesen zeilenweise, da Input kurze Zeilen sein *sollte*
- Davon ausgehen, das zu jedem Block-Anfang *kurze* Zeit später ein Block-Ende kommen *sollte* (Quotes, Klammern...)
- explodierende Rekursionen (zB bei verschachteltem MIME)
- mangelndes Verständnis von Regex

Probleme mit Regex

- Regex sind mächtig
- Regex sind komplex
- sehen oft einfacher aus, als sie es sind
- im Allgemeinen nur mit Blick auf erwarteten Input entwickelt
- Können auf unerwartetem Input "explodieren", auch wenn dieses selbst für das geübte Auge nicht offensichtlich ist
- keine Möglichkeit Resourcen für Regex zu beschränken
- Leider brauchen auch die Kunden die Mächtigkeit der Regex in ACLs, können jedoch noch weniger perl

Probleme mit Regex, Beispiel 1

- Aufgabe:
  Erhöhe Spamlevel, wenn **mindestens** 3 Empfänger in einer To-Zeile (Addr mit Komma getrennt)

```
# To: bla@fasel.com, me@example.org,...
if ( $header =~m{ ^To: ([^,]+,){3,} }xim ) {
    ...
}
```

- Führte zu `Out of memory` bei 100en von Empfaengern, da versucht wird alle zu matchen.
- ein `{3}` statt `{3,}` wäre besser gewesen

Probleme mit Regex, Beispiel 2

- Aufgabe:
  Matche HTML-Tag mit Attributen.

```
# <a href=... title="..".. >
1 m{ <\w+  #Tag
2   (
3     \s+\w+   #Attribute Name
4     ( \s*=\s* (
5       "[^"]*"  # quoted Value
6       |[^"\s]+ # unquoted
7     )?)? # attr|attr=|attr=value
8   )*
9 \s*> }xs;
```

- Bei `<a href=..100e spaces..` wird es exponentiell langsamer, da Regex Engine sich nicht zwischen Matches in Zeile 3 und Zeile 4 entscheiden kann.
- Lösung bestand daran nicht ganz so viel Müll zu akzeptieren

Zusammenfassung

- Vorteile
  - Hohe Produktivität
  - Schnelle Reaktionszeiten
  - Gutes Verhältnis von Code zu Funktionalität, sodaß noch Kraft für Tests und Dokumentation ist
- Nachteile
  - Sicherheitsbewusstsein bei den Maintainern sollte ausgeprägter sein
  - Schwer gute Leute zu bekommen, die wartbaren Code schreiben
  - Regex sind mächtig aber unbeherrschbar
- Wo wären wir ohne perl?
  - Maintainership von IO::Socket::SSL übernommen
  - Net::SIP (VoIP) ist seit kurzem auf CPAN

Ende