Set-versioned package dependencies addressing the problem of shared library updates

Vladimir D. Seleznev

August 2019
## The problem of shared library updates

### Risk of incompatibility between shared libraries and their clients
- Update of a shared library
- Update of a shared library client
- Addition of a new shared library client

### When does it happen in package repositories?
- Update of a shared library

### When does it happen in installed OS?
- OS update from a buggy package repository
- Selective shared library update
- Selective shared library client update
- Installation of a new shared library client
What kind of ABI incompatibilities are problematic

<table>
<thead>
<tr>
<th>ELF level</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Non-versioned symbol removed</td>
</tr>
<tr>
<td>- Versioned symbol removed but the version remains</td>
</tr>
<tr>
<td>- Non-versioned symbol added</td>
</tr>
<tr>
<td>- Versioned symbol added into already existing version</td>
</tr>
<tr>
<td>- Symbol version changed</td>
</tr>
<tr>
<td>- Soname changed</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dwarf level</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Function signature changed in incompatible way</td>
</tr>
<tr>
<td>- Type of variable changed changed in incompatible way</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Function semantics changed</td>
</tr>
</tbody>
</table>
Why these ABI incompatibilities arise

Shared libraries are very easy to produce

- gcc: `-fPIC/-fPIC` to compile, `-shared` to link
- autotools: add `AC_PROG_LIBTOOL` to `configure.ac`, replace `lib_LIBRARIES = libhello.a` with `lib_LTLIBRARIES = libhello.la` and `libhello_a_SOURCES` with `libhello_la_SOURCES` in `Makefile.am`

ABI stability is hard to maintain

- Maintaining ABI requires intelligent design and technical skills
- High entry threshold: part 3 "Maintaining APIs and ABIs" of "How To Write Shared Libraries" takes 8 pages (37K) of technical text
ABI incompatibilities are easy to miss

$ git show 8f6be98bf7b9e9015ad035f34b8414e82c7b68ca
...
@@ -287,7 +287,7 @@
    --prefix=%_prefix  --opensslldir=%_sysconfdir/pki/tls $sslflags \
    --system-ciphers-file=%_sysconfdir/crypto-policies/back-ends/openssl.config \
    zlib sctp enable-camellia enable-seed enable-tlsext enable-rfc3779 \
-   enable-cms enable-md2 \
+   enable-cms enable-md2 enable-ssl2 \
    no-mdc2 no-rc5 no-ec2m no-gost no-srp \n    --with-krb5-flavor=MIT  --enginesdir=%_libdir/openssl/engines \n    --with-krb5-dir=/usr shared $sslarch %?!nofips:fips
@@ -502,6 +502,10 @@
 rm -rf $RPM_BUILD_ROOT/%_libdir/fipscanister.*
 %postun libs -p /sbin/ldconfig

%changelog
+* Wed Mar  2 2016 Tomáš Mráz <tmraz@redhat.com> 1.0.2g-2
+-  reenable SSL2 in the build to avoid ABI break (it does not
+  make the openssl vulnerable to DROWN attack)
+ *
* Tue Mar  1 2016 Tomáš Mráz <tmraz@redhat.com> 1.0.2g-1
- minor upstream release 1.0.2g fixing security issues
What could be done about it

**Classroom**
- Educate students how to write shared libraries

**Package repository**
- Detect and fix incompatible ABI changes early
- Ensure that all libraries being used are linked with
- Check that nothing is indirectly linked with two library versions

**Installed OS**
Check at package installation time that
- every required shared library is provided
- every required shared library version interface is provided
- every ELF symbol required from a shared library is provided by that shared library
### ELF symbol package dependencies

#### Naïve approach 1
- Put into package `Requires` all undefined symbols
- Put into package `Provides` all symbols suitable for resolving undefined symbols

#### Pros
- Similar to the behaviour of dynamic linker

#### Cons
- The size of `Provides` and `Requires` is prohibitively big
- Resolving of these dependencies is prohibitively slow
ELF symbol package dependencies

Naïve approach 2

- Put into package Requires all undefined symbols for each shared library being linked, e.g.
  `libdw Requires: libz.so.1 = [gzclose, gzdopen, gzread, gzerror, gzdirect], ...`
- Put into package Provides all symbols suitable for resolving undefined symbols

Pros

- Similar to the behaviour of dynamic linker
- Resolving of these dependencies is faster than in Naïve approach 1

Cons

- The size of Provides and Requires is still prohibitively big
- Resolving of these dependencies is still very slow
ELF symbol package dependencies

Probabilistic approach

- Put into package Requires **hash values** of all undefined symbols for each shared library being linked instead of strings themselves
- Put into package Provides **hash values** of all symbols suitable for resolving undefined symbols instead of strings themselves

Pros, compared to Naïve approach 2

- The size of Provides and Requires does not depend on symbol length and can be made much smaller
- Resolving of these dependencies can be made much faster

Cons, compared to Naïve approach 2

- False positives
- Error diagnostics does not contain symbol names
Size required to pack arbitrary number of strings

For the given false positive rate $P$:

- **Theoretical minimum:**
  $-\log_2 P$ bits per string,
  for $P = 2^{-10}$ it is **10** bits per string

- **Bloom filter minimum:**
  $-\log_2 P / \log 2 \approx -1.44 \log_2 P$ bits per string,
  for $P = 2^{-10}$ it is $\approx 14.43$ bits per string

- **Set-versions minimum:**
  $\log_2 \left( \frac{N^N}{P} \right) / N = -\log_2 P + \log_2 N - \log_2 N! / N \approx$
  $-\log_2 P + 1 / \log 2 \approx -\log_2 P + 1.44$ bits per string,
  for $P = 2^{-10}$ it is $\approx 11.44$ bits per string

Complexity of $\textbf{Requires} \subset \textbf{Provides}$ test

$O(\|\text{Requires}\| + \|\text{Provides}\|)$
Set-versioned package dependencies

### Hash function
- Based on Jenkins’s one-at-a-time 32-bit hash
- Bitness selected automatically on the number $N$ of strings: $\text{ceil}(\log_2 N - \log_2 P)$
- For the false positive rate $P = 2^{-10} \approx 0.1\%$ the bitness is $\text{ceil}(\log_2 N) + 10$

### Compression
- Hash values are sorted and delta-encoded,
- then compressed using Golomb-Rice coding
- and encoded using Base62 encoding

### Decompression
- Set is decoded using Base62 decoding
- and decompressed using Golomb-Rice coding
$ rpm -q -R -p libdw-0.175-alt1.x86_64.rpm | grep set:
libbz2.so.1()(64bit) >= set:igv0XRQuy1
liblzma.so.5()(64bit) >= set:kiyIz7cr3p0
libz.so.1()(64bit) >= set:khSFaXxmvvC5PDH1

$ rpm -q --provides -p bzlib-1.0.6-alt5.x86_64.rpm \
liblzma-5.2.4-alt1.x86_64.rpm \
zlib-1.2.11-alt1.x86_64.rpm | grep set:
libbz2.so.1()(64bit) = set:idMZep0Zzy6jybSdPuIOCaJZg3s7Tj0ERZ→
lcin9a3qkZsRn56ka0
liblzma.so.5()(64bit) = set:kdZ9NO3hrsVYeAE0SbKiT8b9c84UbuG7B0→
XUj1MZCGedD5YIfxjIAdhzXR5o8Cq9Z6HkvBEDz41TgGNipGBCzEVIqgOg6R→
4HNUpGJZtc8MmMRZciN7oopDEhZ0XJReegsaZJTneJqYA72bwK9GnURtIR4→
mmexdkksaxFHPovnvVxr4tgyEKGhm69bLwhaDjhW6Ac0
libz.so.1()(64bit) = set:kd3mbJvh56uT0sqJBEJ1Zghvez10LBSFJ8Mt→
xDeef1N9vYPZBxcnLIDXTS3wRwf8Z5d2b156CGApsvXkI26pVRfR0YqhzwdhF→
BhPqAvWklha4wsxwNQ7Mc3PyEJU0a99Nq0668JZGLecvY9B0cXVPA0rK2r2z→
EurrbdvRRvG6HbcG1
### Pros
- Guarantees that every ELF symbol required from a shared library is provided by that shared library
- The check is performed in the beginning of every package install/upgrade transaction
- Performance is close to the theoretical maximum

### Cons
- The check is probabilistic
- The check takes time
- Provides and Requires for large libraries look quite big
- Error diagnostics does not contain symbol names
- Base62 is not the most compact ASCII representation, Base85 would save $\frac{1}{3} - \frac{1}{4} = \frac{1}{12}$ of the final representation
Set-versioned package dependencies: obstacles

**Code complexity**

- Based on math and uses relatively uncommon algorithms
- Heavily optimized for performance, it takes efforts and time to understand.
- Some people are unwilling to invest their time into complex things.

The main rpm.org guy once shamelessly admitted that "it's too clever for my taste" and refused to discuss it: [github.com/rpm-software-management/rpm/issues/362#issuecomment-364926552](https://github.com/rpm-software-management/rpm/issues/362#issuecomment-364926552)

**Integration complexity**

- Various projects tend to implement package version checks themselves
- All package depsolvers have to be taught to use version check routines provided by the OS
Set-versioned package dependencies: ideas

**Encoding implementation enhancements**
- Replace Base62 with Base85
- Optimize Golomb Parameter selection in Golomb-Rice coding

**Check ELF symbol version changes**
- Non-versioned undefined symbol can be resolved either to a non-versioned symbol or to a symbol with any version
- Versioned undefined symbol can be resolved either to a non-versioned symbol or to a symbol with the specified version

**Check Dwarf level incompatibilities**
- Use signatures instead of names
- Ignore "insignificant" differences
2004: A. Kiely
Selecting the Golomb Parameter in Rice Coding
ipnpr.jpl.nasa.gov/progress_report/42-159/159E.pdf

2005: Anna Pagh, Rasmus Pagh, S. Srinivasa Rao
An Optimal Bloom Filter Replacement
www.it-c.dk/people/pagh/papers/bloom.pdf

2007: Felix Putze, Peter Sanders, Johannes Singler
Cache-, Hash- and Space-Efficient Bloom Filters
algo2.iti.kit.edu/singler/publications/cacheefficientbloomfilters-wea2007.pdf

2010: Alexey Tourbin
Комплémentарное хеширование подмножеств
ftp.altlinux.org/pub/people/at/protva-2010.pdf

2019: Dmitry Levin
Set-versioned package dependencies
Deployed in ALT:

- set-string, golomb, and base62 routines
  git.altlinux.org/gears/r/..git?p=rpm-build.
  git;a=blob;f=lib/set.c
- mkset tool
  git.altlinux.org/gears/r/..git?p=rpm-build.
  git;a=blob;f=tools/mkset.c
- suggest_bpp tool
  git.altlinux.org/gears/r/..git?p=rpm-build.
  git;a=blob;f=tools/suggest_bpp.c
- provided_symbols tool
  git.altlinux.org/gears/r/..git?p=rpm-build.
  git;a=blob;f=scripts/provided_symbols

Research by Alexey Tourbin:

- rpm set-versions work in progress
  github.com/svpv/rpmss