Friedrich-Alexander-Universität
Faculty of Engineering

Luci: Loader-based Dynamic Software Updates for Off-the-shelf Shared Objects

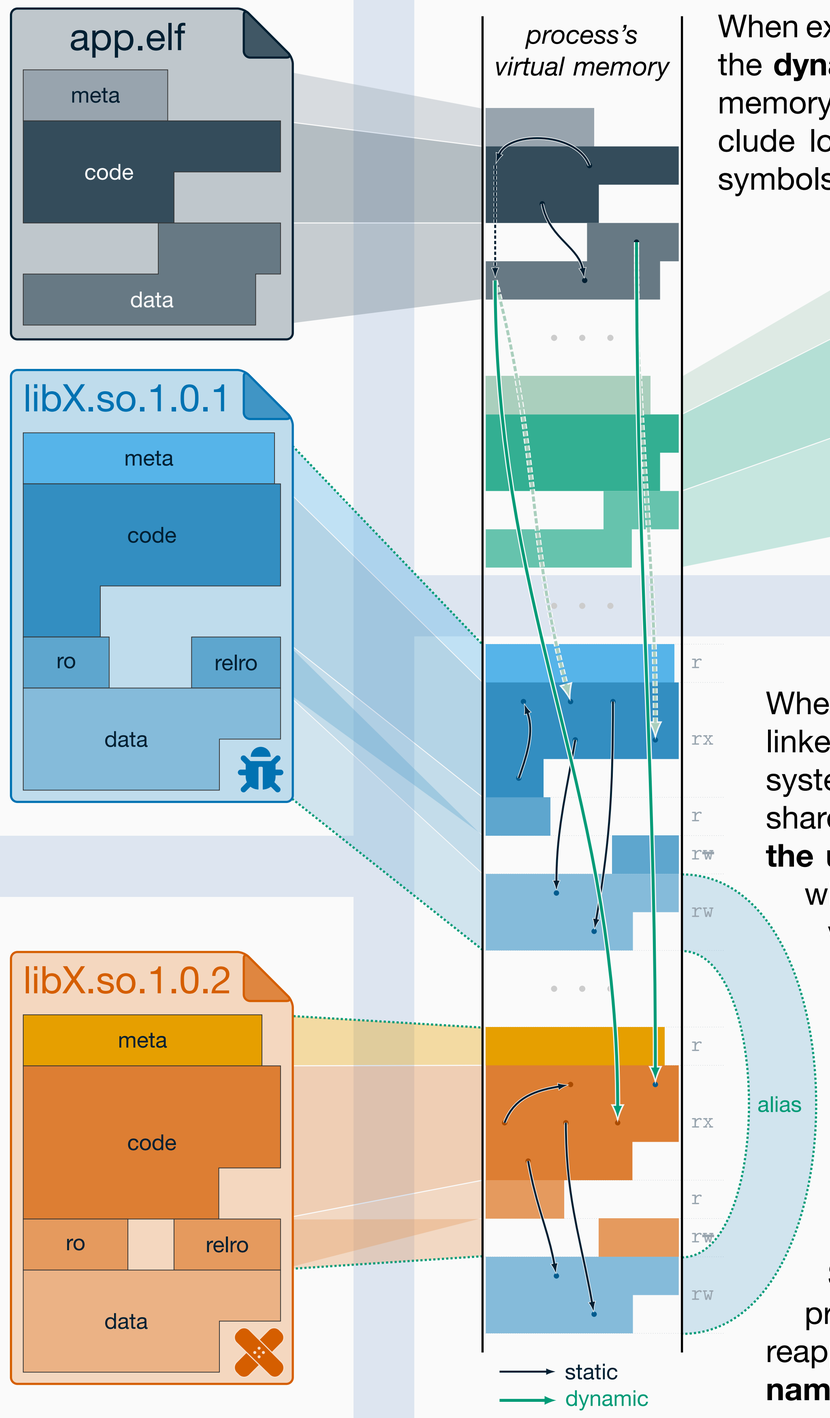
Bernhard Heinloth, Peter Wägemann, Wolfgang Schröder-Preikschat

Problem

Today's system software relies heavily on **dynamically linked libraries**: While this implies especially the standard library, using further third-party libraries for cryptography, compression, or parsing is not uncommon. However, an **application that uses these libraries** will inevitably **inherit their vulnerabilities**.
For example, the *Apache HTTP Server* and its core modules have had **14 vulnerabilities** with high severity since 2010, while the libraries of its six basic dependencies have had over **80 such vulnerabilities** at the same time.
Applying fixes requires a **timely restart** – which is undesirable for stateful software systems or systems with active client connections.
While **dynamic software updates** are a remedy for this, existing approaches **require modifications** to either the source code or the build chain, preventing deployment on a broad scale in user space. Luci addresses this shortcoming with a solution that relies only on **unmodified ELF shared objects** as they are commonly distributed in Unix-like systems.

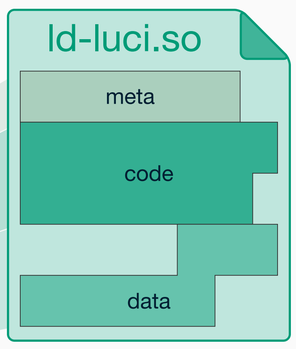
Update Constraints

For the Luci approach, an updated library release must be identical to the previous version in the **layout of the writable data segment**. Furthermore, neither the **interface to the library (API)** nor the **initialization routines** must change.
Since common bugfix patterns only affect the function scope, while structural changes are rare and API & ABI compatibility is maintained, the **requirements are usually met for non-feature updates**.
Using the ELF metadata and, if available, the DWARF debug information, Luci is able to detect changes that are not eligible for dynamic software updates.



Luci Outline

When executing an application, the operating system loads the **dynamic linker/loader** (RTLD / `ld.so`) into the virtual memory of the process. The loader's responsibilities include loading all necessary libraries, binding unresolved symbols, and initializing all components before startup.



Luci is a **drop-in replacement** for the *glibc* loader on the x86_64 architecture, allowing for easy & user-transparent deployment without requiring additional permissions since it already has access to the process's virtual memory.

Luci Workflow

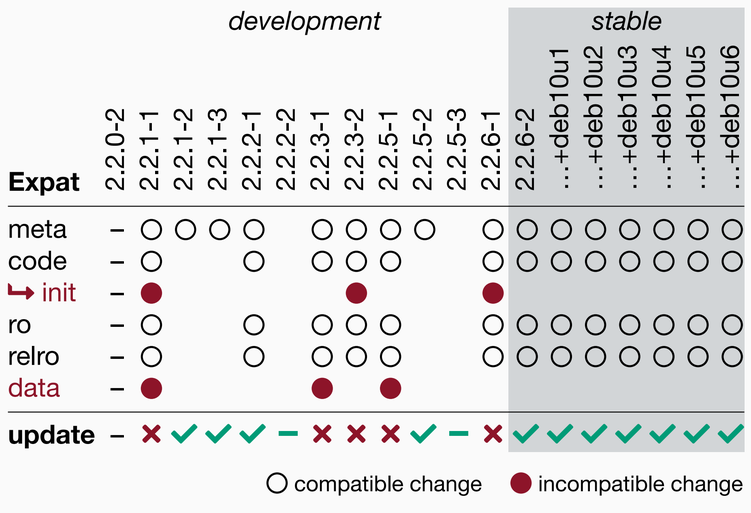
When an application is started with the Luci dynamic linker/loader and a used **libraries changes** on the file system, **Luci automatically detects and analyzes** the shared object. If Luci finds incompatibilities, it **notifies the user** about the need for a manual restart. Otherwise, Luci **loads the updated shared object** into the virtual memory of the running process:
The sections are mapped to a (previously unused) memory area, except for the writable sections, which are instead **memory aliases** to the corresponding sections of the previous version. Then **all references in the process are updated** to point to the new version – similar to lazy binding, only the *Global Offset Table* must be adjusted, and therefore **quiescence is not required**.
Since no new indirections are introduced, this approach imposes **no runtime overhead** and can be reapplied to subsequent releases, allowing **multiple dynamic updates** of the same library.

Results in Off-the-shelf Shared Objects

The Luci approach is tested on several popular library releases from the Debian and Ubuntu Linux distributions. During the execution of test suites (with significant code coverage), the shared objects on the file system are replaced with subsequent releases collected from prebuilt packages in the official repositories.

Expat Library Releases in Debian Buster

Out of 18 updated shared object packages, only five updates – all of them during the development phase of Debian Buster – do not meet the requirements and are, therefore, not eligible for dynamic updates.



Distribution Overview for Expat

For all *Expat* packages in recent distribution versions, Luci maintains the **ability to update most** releases of shared objects without the need to restart the application. **Especially during the stable phase**, while keeping the API and only applying bug fixes, Luci is able to achieve high dynamic update rates.

Build		Expat	update
custom (vanilla)		2.0.0 – 2.5.0	17 / 26 (65%)
Debian Buster	*all*	2.2.0 – 2.2.6	13 / 18 (72%)
	stable	2.2.6	6 / 6 (100%)
Debian Bullseye	*all*	2.2.7 – 2.2.10	9 / 10 (90%)
	stable	2.2.10	5 / 5 (100%)
Ubuntu Focal	*all*	2.2.7 – 2.2.9	6 / 6 (100%)
	stable	2.2.9	4 / 4 (100%)
Ubuntu Jammy	*all*	2.4.1 – 2.4.7	10 / 12 (83%)
	stable	2.4.7	2 / 2 (100%)

Summary of Evaluated Libraries

Evaluation of dynamic updates using Luci, performed on *Apache HTTP Server* core dependency libraries, with prebuilt packages from different Debian and Ubuntu distribution releases:

- ✓ Expat XML parser
- ✓ Extended Crypt Library
- ✗ OpenSSL
- ✓ Perl 5 Compatible Regular Expression Library
- ✓ Zlib

While Luci can apply the **majority of updated releases** without a restart for most libraries, only *OpenSSL* cannot be dynamically updated, mainly due to its extensive **use of function pointers** in global data. However, even in this case, Luci detects the incompatibility and **notifies the user** while continuing to run the application using the old library.

Supported by DFG German Research Foundation
Project number 465958100 — SCHR 603/16-1 "NEON"
Project number 502947440 — WA 5186/1-1 "Watwa"

Code and Artifacts at
github.com/luci-project/eval-atc23