中国科学院软件研究所学术年会 暨重点实验室科技活动周 2025 第十届
学术论文

D-Linker:一种基于Object文件的定制化共享库裁剪方法

何家泰, 侯朋朋 , 于佳耕, 齐冀*, 孙滢, 李丽娟, 赵瑞霖, 武延军

中国科学院软件研究所
jiatai2021@iscas.ac.cn

IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems （TCAD）2024

Motivation

➢ Background
- Shared libraries are widely used in software development to execute third-party functions.
- Size and complexity of shared libraries tend to increase for supporting new features, resulting in shared library bloating.

➢ Issues
- Shared library bloating causes significant storage & memory wastage especially in embedded systems.
- Code bloating in shared libraries also frequently leads to return-oriented programming (ROP) gadgets that can cause security risks.

Conclusion: It is necessary to customize shared library debloating for embedded systems with specific functionalities.

Limitations of SOTAs

➢ Source Code-Level Debloating
- Analyze the source code or IR
- Identify unused functions
- Avoid loading unused functions at runtime

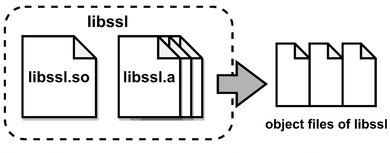
➢ Binary-Level Debloating
- Binary analysis
- Identify unused functions
- Remove unused functions by binary rewriting

	Object-Level Debloating(ours)	Source code-Level Debloating	Binary-Level Debloating
Source Code Needed	No	Yes	No
Runtime Support	No	Yes	No
Gadgets Reduce	Above 50%	Above 80%	Above 50%
Data Debloating	Yes	No	No
File Size Debloating	Above 40%	N/A	Above 20%

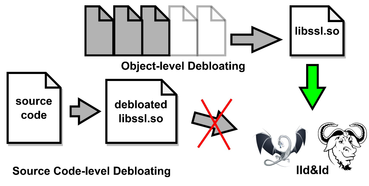
Observation and Insight

Our Insight: object-level de-bloating is especially suitable for embedded system shared library debloating, because it strikes a balance of flexibility and debloating effectiveness.

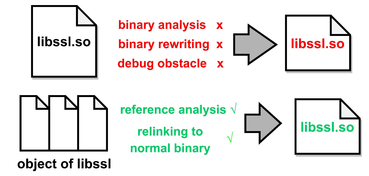
➢ Comparison with Source code-Level Debloating
- It's easier to obtain object files compared to source code



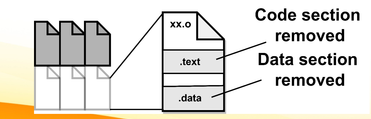
- Object-Level debloating doesn't need runtime support



➢ Comparison with Binary-Level Debloating
- Object-Level debloating has more reliability comparing to binary-level debloating works



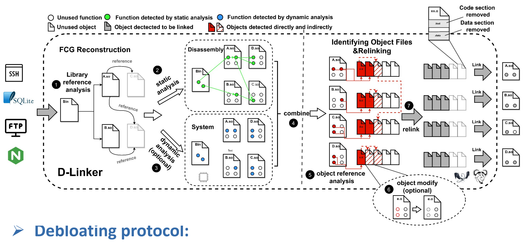
- Object-Level debloating can debloat data sections, making it more efficient in decreasing the size of shared libraries compared to binary-level debloating works



D-Linker's challenges

1. Challenge1: Which Reference should be Removed?
   - It is necessary to identify object files which is referenced but not used
2. Challenge2: How to remove redundant reference?
   - Object files with data dependency can not be safely removed.
   - Some unused object files can not be removed directly due to symbol dependency.

D-Linker's workflow



➢ Debloating protocol:
- Analyzing the reference of object files
- Identifying unused object files
- Generating new shared library without unused object files by relinking

➢ Tow Modes of D-Linker
- **Normal Debloating:**
  - Preserving all dependencies of used object files
  - Absolutely complete
  - Leaving some unused object files
- **In-depth Debloating:**
  - Higher reduction effectiveness
  - Using given test suites
  - Sound correctness with given test suites

➢ Solution1: Redundant dependency analysis
- Classify the dependencies between object files into six categories and identify redundant dependencies.

➢ Solution2: Data dependency analysis

Algo. 1 Get Full Dependency Set and Data Dependency

Require: Object files of a shared library O;
Ensure: full dependency set F and data dependency set

```
function DFSFindDependency(o, dependency)
    if dependency == full then
        F(o).add(o)
        /* x is a object file referenced by o */
        for x ∈ fullreference(o) do
            if x ∉ F(o) then
                DFSFindDependency(x, full)
                F(o).add(F(x))
            end if
        end for
    else if dependency == data then
        D(o).add(o)
        /* x is a object file data referenced by o */
        for x ∈ datareference(o) do
            if x ∉ D(o) then
                DFSFindDependency(x, full)
                D(o).add(D(x))
            end if
        end for
    end if
end function
for o in O do
    DFSFindDependency(o, full)
    DFSFindDependency(o, data)
end for
```

- Obtain the FCG with redundant reference removed
- Categorize the dependencies:
  - data dependencies
  - function dependencies
- Retain two kind of object files:
  - object files in FCG
  - object files data referenced by object files in FCG
- Modify the unused object files and generate debloated shared library

Evaluation

➢ Evaluation Results
- In-depth debloating reduced 44.9% size of shared library of vsftpd， which was widely used in prior works.

Table 1: The Reduction of Library Total Size

Program	File size		
	Baseline	Tailored	Reduction
Nginx			
libc	612KB	169.8KB	-72.3%
libcrypto	1.8MB	265.7KB	-85.3%
libpcre	153.4KB	120.6KB	-21.4%
libz	117KB	59.2KB	-49.5%
Coreutils			
libc	612KB	325KB	-46.9%
Sqlite			
libc	612KB	170.2KB	-72.1%
libz	117KB	88KB	-25.8%
libreadline	379KB	370.6KB	-2.3%
libncurses	398KB	131.1KB	-66.9%
Openssh			
libc	612KB	198.9KB	-67.5%
libcrypto	1.8MB	1.7MB	-5.6%
libz	117KB	96.8KB	-17.3%
Vsftpd			
libc	612KB	166.1KB	-72.9%
libcrypto	1.8MB	1.5MB	-16.7%
libssl	406KB	375.2KB	-7.6%
Vsftpd(In-depth)			
libc	612KB	166.1KB	-72.9%
libcrypto	1.8MB	1.1MB	-38.9%
libssl	406KB	288.2KB	-29.1%
Alpine(In-depth)			
libc	612KB	358.6KB	-41.5%
libcrypto	1.8MB	1.1MB	-38.9%
libssl	406KB	288.2KB	-29.1%

Table 2: The Reduced of Objects in Library

Program	Number of object files		
	Baseline	Tailored	Reduction
Nginx			
libc	1341	366	-72.7%
libcrypto	556	25	-95.5%
libpcre	22	9	-59.1%
libz	16	5	-68.8%
Coreutils			
libc	1341	464	-65.4%
Sqlite			
libc	1341	377	-71.9%
libz	16	10	-37.5%
libreadline	35	33	-5.7%
libncurses	149	31	-79.2%
Openssh			
libc	1341	456	-66.0%
libcrypto	556	404	-27.3%
libz	16	8	-50.0%
Vsftpd			
libc	1341	349	-74.0%
libcrypto	556	390	-29.9%
libssl	46	38	-17.4%
Vsftpd(In-depth)			
libc	1341	349	-74.0%
libcrypto	556	233	-58.1%
libssl	46	24	-47.8%
Alpine(In-depth)			
libc	1341	508	-62.1%
libcrypto	556	233	-58.1%
libssl	46	24	-47.8%

- D-Linker improves debloating effectiveness by 30% compared to binary-level debloating
- incurs a 5% decrease in code gadgets reductioncompared to source-code-level debloating

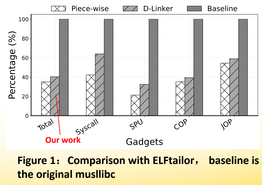


Figure 1：Comparison with ELFtailor， baseline is the original musllibc

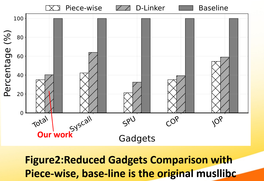


Figure2:Reduced Gadgets Comparison with Piece-wise, base-line is the original musllibc