

libc 软件包分析报告

1. 软件包基本信息

1.1 软件包开发背景与目标

要了解 libc 的背景，首先需要了解一些关于C语言的历史发展。

最初的C语言是在1970年贝尔实验室的Ken Thompson发明的B语言的基础上，由他的同事Dennis M. Ritchie改进而来的。当时的B语言由于没有数据类型，因此不适合用来编写Unix操作系统，Ritchie添加了诸如char，int等数据类型后，在1973年用C语言重写Unix操作系统获得了成功。从此C语言很快成为一门主流的系统编程语言。

后来同在贝尔实验室的Brian W. Kernigan和Dennis M. Ritchie两个人为C语言写了一本经典的教材《The C Programming Language》，对C语言的标准做了详尽的阐述，后来这本书也以这两位作者名字的首字母来指代，被通俗的称为《K&R》。但此时C语言还没有成为一门正式的语言标准。

到1983年，由于Unix已经在很多不同的平台上运行，因此C语言也随之有了更广泛的使用。此时迫切需要制定一个与硬件平台无关的C标准，此后ANSI(American National Standards Institute) 美国国家标准化组织开始着手制定标准，在1989年它发布了第一个关于C语言的标准，官方称之为 ANSI X3.159-1989，通常我们也称之为 ANSI C 标准，也就是我们俗称的“C89”。

为了和之前的C语言区分，人们习惯上把在《K&R》书中所描述的C称之为“K&R C”，这种ANSI C之前的C语言写法，通常没有函数原型。从ANSI C之后，C语言需要有严格的原型声明，以便编译器检查出函数调用时的语法错误和自动进行参数强制类型转换。

1990年，ISO(International Standards Organization)国际标准化组织和IEC(International Electrotechnical Commission)国际电工委员会采纳了ANSI C标准，做了很小的改动后，发布了一个新标准 ISO/IEC 9899:1990，此后ANSI也接受了这一标准作为ANS X3.159-1989的一个替代，这也就是我们俗称的“C90”。后来ISO在1999年又发布了一个修改后的C标准ISO/IEC 9899:1999,就是“C99”。在2011年的时候,这个标准中又增添了5个头文件,被称之为"C11”。

最初的C语言中并没有提供可让用户调用的内部函数，例如I/O(输入/输出)操作。随着C的广泛使用，许多大学和组织开始使用各自不同的函数来编写程序，到了1980年前后，定义一个关于C语言可调用的函数接口标准就成为大家的共同需求。到了1983年，ANSI C标准的制定过程中，同时也开始制定C函数库的标准，这就是 libc 的起因。

libc (C标准库)是C语言所需要用的标准库，最初就是由ANSI标准在C89中定义的，所以也被称为是ANSI C标准库。通常C标准库是和C POSIX库在一起开发的。

POSIX (Portable Operating System Interface)是IEEE定义的可移植操作系统的接口, POSIX不仅定义了应用程序接口, 还包括命令行shell和工具接口, 以确保软件可以在各种Unix版本和其他操作系统上进行移植。最初``POSIX''就是指IEEE Std 1003.1-1988, 这是在1998年IEEE发布的一个标准, 也称之为ISO/IEC 9945。

开始的时候, POSIX就只有一个文档, 包括了核心编程接口。后来逐步发展成为了19个独立文档, 许多用户级别的程序, 服务和工具也成为了标准内容之一。例如awk, echo, ed, 也包括基本I/O服务, 例如file, terminal和network等。到了2009年, POSIX的各个不同的文档被整合为一个标准IEEE Std 1003.1-2008, 也被称为POSIX.1-2008。

在 POSIX.1 中, 大部分都是关于核心服务的, 例如Process进程, Signals信号, File/Directory文件和目录操作, Pipes管道等。其中也包括了C standard library(C标准库)。

任何类Unix操作系统都需要一个C标准库, 因为操作系统和编译器的多样性, libc的实现也有很多种不同的版本。

以下是目前比较流行的 libc 的实现。

- * BSD libc, implementations distributed under BSD operating systems.
- * GNU C Library, used in GNU/Linux and GNU/HURD.
- * Microsoft C Run-time Library, part of Microsoft Visual C++
- * dietlibc, an alternative small implementation of the C standard library (MMU-less)
- * uClibc, a C standard library for embedded Linux systems (MMU-less)
- * Newlib, a C standard library for embedded systems (MMU-less)
- [3]
- * klibc, primarily for booting Linux systems.
- * EGLIBC, variant of glibc for embedded systems.
- * musl, another lightweight C standard library implementation for Linux systems[4]
- * Bionic, originally developed by Google for the Android embedded system operating system,

接下来我们将以GNU libc为例, 具体阐述后面的问题。

GNU libc 是在 GNU 项目中所使用的 C库, 同样也被用在了 linux 中。

GNU libc 项目的主要目标, 主要是为了设计一个可移植的, 高性能的C库。它遵循所有相关标准, 包括ISO C11和POSIX.1-2008。

1.2 软件包维护者信息

长期以来, Ulrich Drepper 一直是 glibc 项目最主要的贡献者和维护者(the lead contributor and maintainer)。他的LinkedIn profile地址在<http://www.linkedin.com/in/ulrichdrepper>

但在2012.5.27发布的一则消息声称，因为开发者社区已经能够自我管理，原先的glibc决策委员会一致同意宣布解散。

glibc项目的方向和决策将会由项目开发者中的积极分子来制定，目前以下人员已经同意对该项目负责。

Ryan Arnold, Maxim Kuvyrkov, Joseph Myers, Carlos O'Donnell, Alexandre Oliva.

从GNU libc 项目主页上看，目前是由一个开发者社区在负责维护，社区中的核心开发人员大约有40多个，大部分的开发人员信息可以从这个页面获得。

<http://sourceware.org/glibc/wiki/MAINTAINERS>

在 LinkedIn 上也有一个开发者的群组(group)，这个群组目前主要是由 Carlos O'Donnell维护。

<http://www.linkedin.com/groups/GLIBC-Developers-4723106>

1.3 软件包所采用的版权协议

glibc 项目是采用 LGPL 协议发布的。

LGPL (GNU Lesser General Public License) 源自于 GPL，但主要为类库使用而设计的。

GPL要求任何使用/修改/衍生之GPL类库的软件必须采用GPL协议，这就使得商业软件不适合用GPL来进行发布。

LGPL允许商业软件通过类库引用(link)方式使用LGPL类库而不需要开源商业软件的代码。

因此LGPL协议的开源代码很适合作为第三方类库被商业软件引用。

1.4 软件包的演化历史

在2.16版本之前的软件包，主要是通过http/ftp的方式下载。http/ftp的下载地址是：

<http://ftp.gnu.org/gnu/glibc/>

在2.16版本发布之后，软件包被放到了git的仓库中，可以通过git clone命令来获得。

```
git clone git://sourceware.org/git/glibc.git
cd glibc
git checkout --track -b local_glibc-2.16 origin/release/2.16/
master
```

从软件包的修改时间来看，目前所能找到的最早的软件包是1994年的glibc-1.09-crypt.tar.gz

glibc-1.09-crypt.tar.gz	07-Nov-1994 03:00	29K
-------------------------	-------------------	-----

最新的软件包是2012年的glibc-2.16.0.tar.xz

glibc-2.16.0.tar.xz	30-Jun-2012 16:07	9.5M
---------------------	-------------------	------

其中相对比较重要的偶数号版本的发行时间

glibc-2.0.1.bin.alpha-linux.tar.gz	04-Feb-1997 03:00	8.3M
glibc-2.2.1.tar.gz	13-Jan-2001 16:18	14M
glibc-2.4.tar.gz	06-Mar-2006 07:30	20M
glibc-2.6.tar.gz	17-May-2007 22:24	20M
glibc-2.8.tar.gz	10-Mar-2009 17:05	20M
glibc-2.10.1.tar.gz	17-May-2009 21:27	21M
glibc-2.12.1.tar.gz	03-Aug-2010 06:09	20M
glibc-2.16.0.tar.xz	30-Jun-2012 16:07	9.5M

它的0.1-0.6版最初发布的时间大约是在1991.10--1992.2，在 1992.2 发布了第一个正式版本 1.0。

从glibc 2.5版本之后，有一个重要的分支开始被创建出来，它就是 eglibc，也就是 Embedded GLIBC。

这是专为在嵌入式系统使用而设计的，主要是为了glibc能够更好的被交叉编译到各种不同的硬件系统上，因此更注重裁剪性和可移植性。

目前这个项目分支拥有自己单独的主页 <http://www.eglibc.org/home>，当前的最新版本是2.16版，它也是以LGPL协议方式发布的。

2009年5月之后，Debian开始弃用glibc而改用EGLIBC，这也影响到了其后的Ubuntu发行版上，Ubuntu所使用的也是 EGLIBC，。

通过执行这个文件 `/lib/libc.so.6`，就可以看出当前安装的 glibc 的版本。或者也可以运行 `ldd --version` 查看当前版本。

```
$ /lib/libc.so.6
```

```
GNU C Library (Ubuntu EGLIBC 2.11.1-0ubuntu7.8) stable release version 2.11.1, by Roland McGrath and Ulrich Drepper  
Copyright (C) 2009 Free Software Foundation, Inc.
```

```
This is free software; see the source for copying conditions.
```

```
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

```
Compiled by GNU CC version 4.4.3.
```

```
Compiled on a Linux >>2.6.24-27-server<< system on 2011-01-21.
```

```
Available extensions:
```

```
crypt add-on version 2.1 by Michael Glad and others
```

```
GNU Libidn by Simon Josefsson
```

```
Native POSIX Threads Library by Ulrich Drepper et al
```

```
BIND-8.2.3-T5B
```

```
For bug reporting instructions, please see:
```

```
<http://www.debian.org/Bugs/>.
```

```
$ ldd --version
```

```
ldd (Ubuntu EGLIBC 2.11.1-0ubuntu7.8) 2.11.1
```

```
(C) 2009 Free Software Foundation, Inc.
```

Roland McGrath Ulrich Drepper

1.5 类似的可替代软件包简介overseen

uClibc

- uClibc 是一个面向嵌入式Linux系统的小型C标准库。最初uClibc是为了支持uClinux而开发，因为uClinux不支持内存管理单元MMU，因此uClibc也适合于无MMU的硬件系统上。项目的维护者是Erik Andersen，许可证遵从GNU LGPL。2012年5月15日，发布了最新版本uClibc-0.9.33.2。
- 项目主页 <http://www.uclibc.org/>

Newlib

- Newlib 是目前Cygwin所使用的C标准库，是由Cygwin Solutions公司负责开发，在1999年11月15日，Cygwin Solutions宣布将与红帽公司Redhat合并，因此Newlib的主要维护者也是Redhat公司的员工-Corinna Vinschen和Jeff Johnston。Newlib由libc和libm两个库组成，特点是轻量级，速度快，可移植到很多CPU结构上。该项目2011年12月19日发布了最后一个版本1.20.0。
- 项目主页 <http://www.sourceware.org/newlib/>

BSD libc

- BSD libc 是 FreeBSD 操作系统所使用的 C 标准库，采用 BSD 协议发布。2012 年 11 月 17 日发布了最新版本 1.9.2.3。
- 项目主页 <http://www.freebsd.org/cgi/cvsweb.cgi/src/lib/libc/>

Bionic libc

- Bionic libc 是从 BSD 标准 C 库发展而来，以 BSD 许可形式开源。它是由 Google 公司为其 Android 嵌入式手机平台开发的标准库。它的设计目标是在 Linux 之上采用 BSD 协议，实现更快执行速度和更小代码体积的 C 库。

dietlibc

- dietlibc 是一个非常精简的 C 标准库实现。发起者和创始人是德国的 Felix von Leitner，其目标是能够将编译链接之后生成的可执行代码减到最少。这个项目和之前的 libc 项目没有什么关联，完全是从头写起的。它并没有实现 glibc 中的所有函数，只实现了最常用的部分。采用 GPL2 协议发布。
- 项目主页 <http://www.fefe.de/dietlibc/>

EGLIBC

- EGLIBC 是 GNU C 标准库 (glibc) 的一个分支，主要是为嵌入式设备进行优化。它采用 LGPL 协议发布。
- 项目主页 <http://www.eglibc.org/>

1.6 软件包与可替代软件包对比分析

对比 应用领域 采用协议 特点 glibc 桌面领域，GNU/Linux 操作系统 LGPL 功能全面 uClibc 嵌入式领域，uCLinux 操作系统 LGPL 主要用在 uCLinux 系统中 Newlib 桌面领域，Cygwin 系统 LGPL 具有独特的设计，可移植性强 BSD libc 桌面领域，FreeBSD 操作系统 BSD 主要用在 BSD 系统中 Bionic libc 手机嵌入式领域，Android 操作系统 BSD 主要用在 Android 系统中 dietlibc 各种对体积尺寸要求严格的领域 GPL2 设计非常精巧，适合学习 EGLIBC 桌面领域，Ubuntu 操作系统 LGPL 主要用在 Ubuntu 系统中

2. 软件包与发行版的关系

本小节需要介绍和阐述软件包在不同发行版中的名称、功能、运行环境的异同，除了正文描述以外，最好填下表以便更清晰，建议不超过1页。

发行版名称及版本号	软件包在该发行版中的名称	该软件包的版本	该软件包的安装
Ubuntu 10.04	EGLIBC	2.11.1	<code>sudo apt-get install libc6-dev</code>
Redhat9	glibc	2.3.2	<code>rpm -Uvh glibc-kernheaders-2.4-8.34.i386.rpm rpm -Uvh glibc-headers-2.3.2-95.3.i386.rpm rpm -Uvh glibc-devel-2.3.2-95.3.i386.rpm</code>
Fedora	glibc	2.15-23.fc17	<code>yum install glibc-devel.i686 glibc.i686</code>
OpenSUSE	glibc	2.3.3	<code>rpm -ivh glibc-devel-2.3.3-98.28.i586.rpm</code>

3. 软件包的功能、设计架构和接口使用说明

3.1 软件包的功能说明

glibc 是Linux系统中最底层的库函数接口，其他的运行库都会依赖于这个库。它除了封装了Linux系统调用，同时也提供了一些其他必要功能，主要包括：

- 字符串处理 `string`
- 信号处理 `signal`
- 本地化 `locale`
- 数学库 `math`
- 线程库 `linuxthreads`
- 动态内存管理 `malloc`
- 动态加载器 `elf`
- C 标准库 `libc`

3.2 软件包的设计架构

1

到目前为止，C标准库规定了29个头文件，如下：

C89 C89标准规定了15个头文件，分别是 `assert.h` `ctype.h` `errno.h` `float.h` `limits.h` `locale.h` `math.h` `setjmp.h` `signal.h` `stdarg.h` `std-def.h` `stdio.h` `stdlib.h` `string.h` `time.h`

NA1 在95年的修正版NA1中，增补了3个头文件，分别是 `iso646.h`, `wchar.h`, and `wctype.h`

C99 C99标准增加了6个新的头文件，分别是 `complex.h`, `fenv.h`, `inttypes.h`, `stdbool.h`, `stdint.h`, and `tgmath.h`

C11 C99标准增加了5个新的头文件，分别是 `stdalign.h`, `stdatomic.h`, `stdnoreturn.h`, `threads.h`, and `uchar.h`

3.3 软件包的接口说明

3 2-3

因为主要讨论libc，因此对于glibc中不在libc范畴内的不在此处涉及。

输入输出 `stdio.h`

文件操作函数

```
FILE * fopen (const char *filename, const char *opentype)
int fclose (FILE *stream)
int fflush (FILE *stream)
```

格式化输出函数

```
int printf (const char *template, . . . )
int fprintf (FILE *stream, const char *template, . . . )
int sprintf (char *s, const char *template, . . . )
```

格式化输入函数

```
int scanf (const char *template, . . . )
int fscanf (FILE *stream, const char *template, . . . )
int sscanf (const char *s, const char *template, . . . )
```


字符输入输出函数

```
int fputc (int c, FILE *stream)
int fputs (const char *s, FILE *stream)
int putc (int c, FILE *stream)
int putchar (int c)
int puts (const char *s)

int fgetc (FILE *stream)
char * fgets (char *s, int count, FILE *stream)
int getc (FILE *stream)
int getchar (void)
char * gets (char *s)
ssize_t getline (char **lineptr, size_t *n, FILE *stream)

int ungetc (int c, FILE *stream)
```

文件读写函数

```
size_t fread (void *data, size_t size, size_t count, FILE *stream)
size_t fwrite (const void *data, size_t size, size_t count, FILE *stream)
```

文件定位函数

```
int fseek (FILE *stream, long int offset, int whence)
long int ftell (FILE *stream)
void rewind (FILE *stream)
```

错误处理函数

```
int feof (FILE *stream)
int ferror (FILE *stream)
```

字符类别测试 ctype.h

```
int islower (int c)
int isupper (int c)

int isalpha (int c)
int isdigit (int c)
int isalnum (int c)
```

```
int isxdigit (int c)
```

```
int isspace (int c)
```

```
int isblank (int c)
```

```
int isascii (int c)
```

```
int tolower (int c)
```

```
int toupper (int c)
```

```
int toascii (int c)
```

字符串函数 string.h

```
char * strcpy (char *restrict to, const char *restrict from)
```

```
char * strncpy (char *restrict to, const char *restrict from, size_t size)
```

```
char * strcat (char *restrict to, const char *restrict from)
```

```
char * strncat (char *restrict to, const char *restrict from, size_t size)
```

```
int strcmp (const char *s1, const char *s2)
```

```
int strncmp (const char *s1, const char *s2, size_t size)
```

```
char * strchr (const char *string, int c)
```

```
char * strrchr (const char *string, int c)
```

```
size_t strspn (const char *string, const char *skipset)
```

```
size_t strcspn (const char *string, const char *stopset)
```

```
char * strstr (const char *haystack, const char *needle)
```

```
size_t strlen (const char *s)
```

```
size_t strnlen (const char *s, size_t maxlen)
```

```
char * strerror (int errnum)
```

```
char * strtok (char *restrict newstring, const char *restrict delimiters)
```

```
char * strtok_r (char *newstring, const char *delimiters, char **save_ptr)
```

```
char * strsep (char **string_ptr, const char *delimiter)
```

```
char * strpbrk (const char *string, const char *stopset)
```

数学函数 math.h

```
double sin (double x)
```

```

double cos (double x)
double tan (double x)
double asin (double x)
double acos (double x)
double atan (double x)
double atan2 (double y, double x)

double sinh (double x)
double cosh (double x)
double tanh (double x)

double exp (double x)
double pow10 (double x)

double log (double x)
double log10 (double x)
double log2 (double x)

double sqrt (double x)

double ceil (double x)
double floor (double x)

double fabs (double number)

double frexp (double value, int *exponent)

double modf (double value, double *integer-part)
double fmod (double numerator, double denominator)

```

实用函数 stdlib.h

```

double atof (const char *string)

int atoi (const char *string)
long int atol (const char *string)

long int strtol (const char *restrict string, char **restrict tailptr, int base)
double strtod (const char *restrict string, char **restrict tailptr)
float strttof (const char *string, char **tailptr)

int rand (void)
void srand (unsigned int seed)

```

```
void * malloc (size_t size)
void * calloc (size_t count, size_t eltsize)
void *realloc (void *addr, size_t size)
void free (void *ptr)
```

```
void abort (void)
void exit (int status)
int atexit (void (*function) (void))
```

```
int system (const char *command)
char * getenv (const char *name)
int putenv (char *string)
```

```
void * bsearch (const void *key, const void *array, size_t count, size_t size, comparison_fn_t compare)
void qsort (void *array, size_t count, size_t size, comparison_fn_t compare)
```

```
int abs (int number)
long abs (long number)
```

```
div_t div (int numerator, int denominator)
ldiv_t ldiv (long int numerator, long int denominator)
```

诊断 assert.h

```
void assert (int expression)
```

可变参数表 stdarg.h

```
va_list
void va_start (va_list ap, last-required)
type va_arg (va_list ap, type)
void va_end (va_list ap)
```

非局部跳转 setjmp.h

```
int setjmp (jmp_buf state)
void longjmp (jmp_buf state, int value)
```

信号 signal.h

```
sighandler_t signal (int signum, sighandler_t action)
int raise (int signum)
```

日期与时间 time.h

```
time_t time (time_t *result)

struct tm {}
struct tm * localtime (const time_t *time)
```

具体实现相关 float.h limits.h

```
#define SCHAR_MAX    127
#define UCHAR_MAX    255
#define SHRT_MAX     32767
#define USHRT_MAX    65535
```

errno.h

```
volatile int errno
void error (int status, int errnum, const char *format, . . . )
```

locale.h

```
char * setlocale (int category, const char *locale)
struct lconv * localeconv (void)
```

stddef.h

```
typedef long size_t
#define NULL ((void *)0)
```

其它头文件略

```
* iso646.h
* wchar.h
* wctype.h

* complex.h
* fenv.h
* inttypes.h
* stdbool.h
* stdint.h
```

```
* tgmth.h

* stdalign.h
* stdatomic.h
* stdnoreturn.h
* threads.h
* uchar.h
```

4. 软件包漏洞分析

这部分任务书的要求是“针对子任务Linux、Android操作系统安全漏洞检测”中发现的通用基础软件包安全漏洞的确认分析，包括漏洞产生的原因、漏洞可重现条件及相应的测试用例，并可进行复现和验证；“，但是此部分待定，暂时不要去写，将根据后续情况经过大家的讨论之后再写。因为漏洞的检测有赖另外一个项目（“Linux、Android操作系统安全漏洞检测”）给出，如果漏洞非常多，我们可能需要分出重要性，分类进行处理。

5. 软件包的依赖关系

这部分请用文字和图说明该软件包的运行还需依赖哪些其他的软件包，其依赖关系不仅要通过阅读文档、代码来获得，更要通过实际的运行验证来获得依赖关系的证据，这一点请慎重处理。

- Glibc 安装依赖关系
Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Gettext, Grep, Make, Perl, Sed, Texinfo

6. 软件包安全性及特别需要说明的问题

本部分请撰写不能包括在上述章节中的其他重要分析成果，特别是和系统安全有关的内容，如果没有请填写“无”。

- 无

7. 软件包分析成果验证方法

根据任务书，“分析成果进行验证，包括但不限于制作安装包安装测试验证依赖关系、软件包接口测试脚本、可替代软件包兼容性测试验证等。”，具体如何验证后续还有待通过研讨会大家现成讨论确定，但需要在此小节给出进行验证的具体方法和流程的描述，此部分后续在写，暂时空着即可。

- 待定

8. 参考资料

GNU libc 项目主页：<http://www.gnu.org/software/libc/>

GNU libc 参考手册：<http://www.gnu.org/software/libc/manual/>
<http://www.gnu.org/software/libc/manual/pdf/libc.pdf>

libc 说明：http://en.wikipedia.org/wiki/C_standard_library

C STANDARDS (C标准)：<http://www.keil.com/support/docs/1893.htm>

GNU C Library：http://en.wikipedia.org/wiki/GNU_C_Library

Changes in glibc development：<http://lwn.net/Articles/488778/>

开源软件协议比较：<http://www.awflasher.com/blog/archives/939>

GNU C Library version 1.16 发布声明：<http://thread.gmane.org/gmane.comp.lib.glibc.alpha/23457>

C99标准：<http://www.open-std.org/jtc1/sc22/wg14/www/standards.html#9899>