

HOW TO GET STARTED IN HACKING OPENJDK?

Vineel Kumar Reddy Kovvuri
<http://vineelkumarreddy.com>

Contents

1	Introduction	2
2	How do I get the source code?	2
3	How do I build it?	3
4	How do I hack it?	5

1 Introduction

This blog post is not about internals of JDK(Java Development Kit), But a mere documentation for people who want to get started. JDK is open sourced a long time ago. The reference implementation of Java is now based on [OpenJDK](#). So any one interested in it can get the source code and play with it. Here on, when I refer to JDK it means all the components including Java class libraries/JVM(Hotspot)/Java Compiler. In this post, I will walk you in getting and building the latest JDK 9 sources.

2 How do I get the source code?

All the components of JDK 9 are from <http://hg.openjdk.java.net/jdk9/> . OpenJDK is under Mercurial. For any one familiar with Git, this is no different. Below is the layout of repositories in JDK 9 OpenJDK forest.

1. dev - top-level umbrella project for all the sub projects
 - (a) corba - Not my interest
 - (b) hotspot - The Java Virtual Machine implementation. JIT/GC go here
 - (c) jaxp - Not my interest
 - (d) jaxws - Not my interest
 - (e) jdk - The Java class libraries including native JNI implementation go here
 - (f) langtools - The Java compiler and other tools go here
 - (g) nashorn - Not my interest

There are multiple ways to get the code. The easiest being, first clone the umbrella 'dev' project and run `get_source.sh` to pull all other repositories.

```
hg clone http://hg.openjdk.java.net/jdk9/dev 9dev
cd 9dev
sh ./get_source.sh
```

Once done you should see below directory structure.

```
vineel@Z97X:~/9dev$ tree -L 1
.
├── [1.5K] ASSEMBLY_EXCEPTION
├── [4.0K] build
├── [4.0K] common
├── [1.6K] configure
├── [4.0K] corba
├── [3.0K] get_source.sh
├── [4.0K] hacking
├── [4.0K] .hg
├── [ 103] .hgignore
├── [ 17K] .hgtags
├── [4.0K] hotspot
├── [4.0K] jaxp
├── [4.0K] jaxws
├── [4.0K] .jcheck
├── [4.0K] jdk
├── [4.0K] langtools
├── [ 19K] LICENSE
├── [4.0K] make
├── [2.7K] Makefile
├── [ 53K] modules.xml
├── [4.0K] nashorn
├── [1.5K] README
├── [ 57K] README-builds.html
├── [ 50K] README-builds.md
├── [4.0K] test
└── [174K] THIRD_PARTY_README
```

Figure 1

3 How do I build it?

Building OpenJDK is straight forward too, The only precursor is to get the dependencies. The OpenJDKs Adopt OpenJDK project has already documented the dependencies here, Its a matter of sudo apt-get install . Run below commands to build the source code.

bash ./configure #Setup the environment
make all #build the entire forest

```
vineel@Z97X:~/9dev$ tree -L 2 build/
build/
├── [4.0K] linux-x86_64-normal-server-release → Generated by bash ./configure
├── [2.7K] bootcycle-spec.gmk
├── [2.0K] build.log
├── [ 24K] build.log.old
├── [4.0K] buildtools
├── [3.1K] compare.sh
├── [ 11K] configure.log
├── [ 11K] configure.log.old
├── [4.0K] configure-support
├── [4.0K] hotspot
├── [4.5K] hotspot-spec.gmk
├── [4.0K] images → Generated by make all
├── [4.0K] jdk
├── [1.2K] Makefile
├── [4.0K] make-support
├── [ 26K] spec.gmk
└── [4.0K] support
```

Figure 2: OpenJDK build directory after running bash ./configure

```
vineel@Z97X:~/9dev$ tree -L 2 build/*/images
build/linux-x86_64-normal-server-release/images
├── [4.0K] docs
│   ├── [4.0K] api
│   ├── [4.0K] jdk
│   ├── [4.0K] jre
│   └── [4.0K] platform
├── [4.0K] jdk
│   ├── [1.5K] ASSEMBLY_EXCEPTION
│   ├── [4.0K] bin
│   ├── [4.0K] conf
│   ├── [4.0K] demo
│   ├── [4.0K] include
│   ├── [118K] jrt-fs.jar
│   ├── [4.0K] lib
│   ├── [ 19K] LICENSE
│   ├── [4.0K] man
│   ├── [1.2K] release
│   ├── [4.0K] sample
│   ├── [ 51M] src.zip
│   └── [174K] THIRD_PARTY_README
├── [4.0K] jre
│   ├── [1.5K] ASSEMBLY_EXCEPTION
│   ├── [4.0K] bin
│   ├── [4.0K] conf
│   ├── [4.0K] lib
│   ├── [ 19K] LICENSE
│   ├── [4.0K] man
│   ├── [1.0K] release
│   └── [174K] THIRD_PARTY_README
├── [1.4M] sec-bin.zip
├── [4.0K] test
│   ├── [4.0K] hotspot
│   ├── [4.0K] jdk
│   └── [ 15] Readme.txt
└── [ 610] _the.nashorn.jar.vardeps
```

The final **JDK** directory

The final **JRE** directory

Figure 3: OpenJDK images directory after running make all

4 How do I hack it?

Now its all yours! This is where the fun begins, Even though OpenJDK sources are highly organized, you might need the help of some tools to get around with it. My favorite tool of choice for any source code exploration is OpenGrok, I have my local OpenGrok instance setup.



Figure 4

Since images/jdk/bin/java is the binary which executes your class files. I wanted to locate the code for it, so that, I can see where the rabbit hole is leading to(essentially in to hotspot) :, The right way to find out is to decipher the build system to understand how things are getting build and from where they are getting build. I really really! dont want to explore the build system now. So, Its time to wear my black hat and roll up some reverse engineering skills!

With the help of objdump I disassembled the jdk/bin/java binary and looked for symbols, specifically in its main function. Once I know some candidate symbols, I can get to the code from OpenGrok. So below is the disassembled view of main function of the java binary. This function is in turn making calls to JLI_* apis. So I selected JLI_PreprocessArg as my candidate symbol.

```

vineel@Z97X:~/9dev$ objdump -d build/*/images/jdk/bin/java | grep -i -A30 "<main>"
0000000000400880: <main>:
400880: 55                push    %rbp
400881: 48 89 e5          mov     %rsp,%rbp
400884: 41 57             push    %r15
400886: 41 56             push    %r14
400888: 41 55             push    %r13
40088a: 41 54             push    %r12
40088c: 49 89 f6          mov     %rsi,%r14
40088f: 53                push    %rbx
400890: 31 f6             xor     %esi,%esi
400892: 89 fb             mov     %edi,%ebx
400894: bf 01 00 00 00    mov     $0x1,%edi
400899: 48 83 ec 18       sub     $0x18,%rsp
40089d: e8 3e ff ff ff    callq   4007e0 <JLI_InitArgProcessing@plt>
4008a2: 8d 7b 01          lea     0x1(%rbx),%edi
4008a5: 48 63 ff          movslq  %edi,%rdi
4008a8: e8 63 ff ff ff    callq   400810 <JLI_List_new@plt>
4008ad: 85 db             test    %ebx,%ebx
4008af: 49 89 c5          mov     %rax,%r13
4008b2: 7e 6e             jle     400922 <main+0xa2>
4008b4: 8d 43 ff          lea     -0x1(%rbx),%eax
4008b7: 49 8d 44 c6 08    lea     0x8(%r14,%rax,8),%rax
4008bc: 48 89 45 c8       mov     %rax,-0x38(%rbp)
4008c0: 49 8b 3e          mov     (%r14),%rdi
4008c3: e8 28 ff ff ff    callq   4007f0 <JLI_PreprocessArg@plt>
4008c8: 48 85 c0          test    %rax,%rax
4008cb: 49 89 c7          mov     %rax,%r15
4008ce: 0f 84 bd 00 00 00 je      400991 <main+0x111>
4008d4: 48 8b 50 08       mov     0x8(%rax),%rdx
4008d8: 85 d2             test    %edx,%edx
4008da: 7e 2c             jle     400908 <main+0x88>
vineel@Z97X:~/9dev$

```

Disassembly of *main* function in *images/jdk/bin/java* binary

This symbol looks promising!

Figure 5

Now searching for the symbol `JLI_PreprocessArg` in OpenGrok found the source code for `jdk/bin/java`!

Search - Mozilla Firefox

File Edit View History Bookmarks Tools Help

{0 Search x +

localhost:8080/source/search?q=&defs=&refs= Search

{OpenGrok

Home Sort by: last modified time | **relevance** | path

Full Search

Definition

Symbol **JLI_PreprocessArg**

File Path

History

Type

Search **Clear** **Help**

In Project(s) **select all** **invert selection**

- common
- corba
- hotspot
- jaxp
- jaxws
- jdk

Searched **refs:JLI_PreprocessArg** (Results **1 - 4** of **4**) sorted by relevance

/jdk/src/java.base/share/native/launcher/

HAD **main.c** 131 JLI_List argsInFile = **JLI_PreprocessArg**(argv[i]);

/jdk/src/java.base/share/native/libjli/

HAD **jli_util.h** 137 JLI_List **JLI_PreprocessArg**(const char *arg);


HAD **args.c** 369 JLI_List **JLI_PreprocessArg**(const char *arg) **function**

522 JLI_List tokens = **JLI_PreprocessArg**(argv[i]);

/jdk/src/java.base/windows/native/libjli/

HAD **cmdtoargs.c** 288 argsInFile = **JLI_PreprocessArg**(arg);

Completed in 349 milliseconds

served by {OpenGrok on 

Indexes created Mon Mar 28 19:24:16 IST 2016

Figure 6

```

94 main(int argc, char **argv)
95 {
96     int margc;
97     char** margv;
98     const jboolean const_javaw = JNI_FALSE;
99 #endif /* JAVAW */
100     JLI_InitArgProcessing(!HAS_JAVA_ARGS, const_disable_argfile);
101
102 #ifdef _WIN32
103 {
104     int i = 0;
105     if (getenv(JLDEBUG_ENV_ENTRY) != NULL) {
106         printf("Windows original main args:\n");
107         for (i = 0 ; i < _argc ; i++) {
108             printf("wwwd_args[%d] = %s\n", i, __argv[i]);
109         }
110     }
111 }
112 JLI_CmdToArgs(GetCommandLine());
113 margc = JLI_GetStdArgc();
114 // add one more to mark the end
115 margv = (char **)JLI_MemAlloc((margc + 1) * (sizeof(char *)));
116 {
117     int i = 0;
118     StdArg *stdargs = JLI_GetStdArgs();
119     for (i = 0 ; i < margc ; i++) {
120         margv[i] = stdargs[i].arg;
121     }
122     margv[i] = NULL;
123 }
124 #else /* *NIXES */
125 {
126     // accommodate the NULL at the end
127     JLI_List args = JLI_List_new(argc + 1);
128     int i = 0;
129     for (i = 0; i < argc; i++) {
130         JLI_List argsInFile = JLI_PreprocessArg(argv[i]);
131         if (NULL == argsInFile) {
132             JLI_List_add(args, JLI_StringDup(argv[i]));
133         } else {
134             int cnt, idx;
135             cnt = argsInFile->size;
136             for (idx = 0; idx < cnt; idx++) {
137                 JLI_List_add(args, argsInFile->elements[idx]);
138             }
139             // Shallow free, we reuse the string to avoid copy
140             JLI_MemFree(argsInFile->elements);
141             JLI_MemFree(argsInFile);
142         }
143     }
144     margc = args->size;
145     // add the NULL pointer at argv[argc]
146     JLI_List_add(args, NULL);
147     margv = args->elements;
148 }
149 #endif /* WIN32 */
150 return JLI_Launch(margc, margv,
151                 sizeof(const_jargs) / sizeof(char *), const_jargs,
152                 sizeof(const_appclasspath) / sizeof(char *), const_appclasspath,
153                 VERSION_STRING,
154                 DOT_VERSION,
155                 (const_progname != NULL) ? const_progname : *margv,
156

```

Figure 7: Inside main.c

To double confirm, I have added a print message and did the make all and it indeed worked as expected! You could see my print when I execute java from the command line in the right side window.

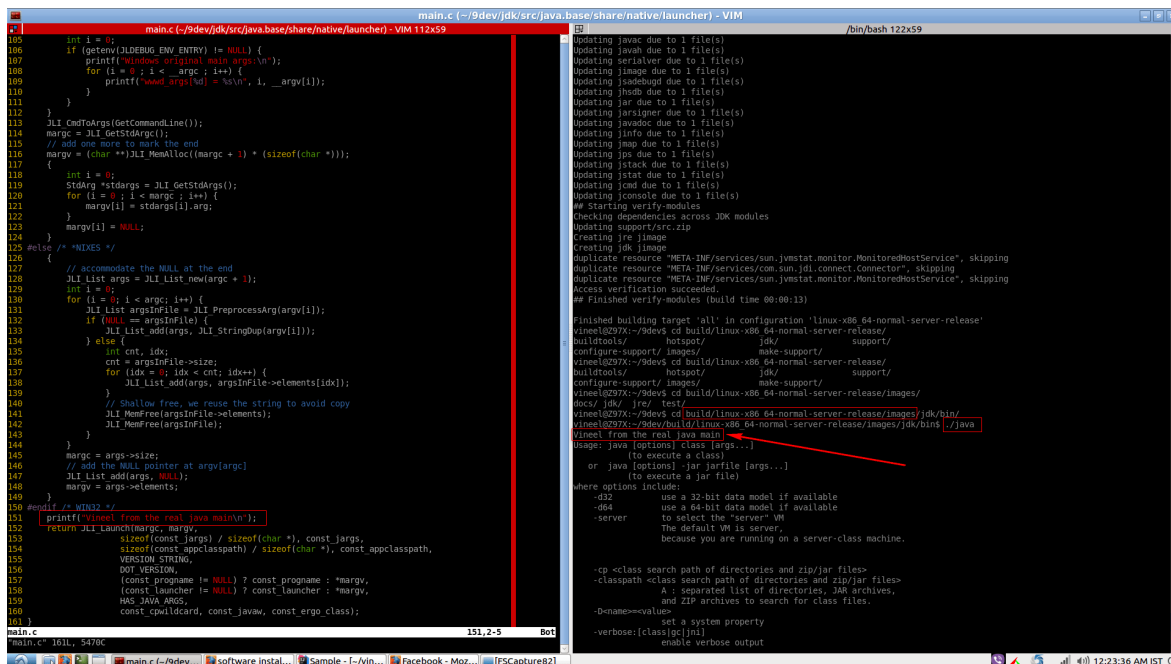


Figure 8: Inside main.c

5 References

1. The OpenJDK Developers' Guide - <http://openjdk.java.net/guide/>