ENDIAN CHECKING PORTABILITY TOOL (ECT)

IBM

Getting Started Tutorial

COPYRIGHT

INTRODUCTION

The Endian Check Tool (ECT) is similar in execution and principle to the "lint" code analysis tool, which is well-known to C developers. The tool performs an analysis of the application binary and sources, looking for code with potential chip architecture dependencies and problems in cross-hardware portability.

This tutorial takes the user through the basic steps to set up ECT, modify the build process, and analyze the results.

Overview

This tutorial will take you through all the steps necessary to test an application for potential endian problems. In the following steps you will install ECT, modify the test application's build process, check the application and interpret the results.

Prior to running through this tutorial you should verify that you have the following packages installed on your system:

SFWgawk	gawk - pattern scanning and processing language
SFWgbin	binutils - GNU binary utilities
SFWgcc	gcc - GNU Compiler Collection
SFWgcc33	gcc-3.3.2 - GNU Compiler Collection
SFWgcmn	gcmn - Common GNU package
SFWgdb	gdb - GNU source level debugger
SFWgmake	make - GNU make utility

Enter "`pkginfo | grep SFW`" at the command line to verify this. For more information, please check the installation instructions in the ECT section of the MSLK manual.

Step 1 - Installation

For the purpose of this tutorial, you will be using the tarball installation method and installing ECT into your home directory. We recommend you don't get fancy with where you put things at this point. You can always clean up the ECT directory and re-install when the tutorial is finished.

Get the ect-1.0.tar.gz file and place it in your home directory (example starting in the directory where the ect-1.0.tar.gz file resides. If this is your home directory skip to the next step).

```
cp ect-1.0.tar.gz ~/
```

Change to your home directory, expand the archive and then extract the files from the tarball:

```
cd
gunzip ect-1.0.tar.gz
tar xvf ect-1.0.tar
```

You will now have an ect subdirectory in your home directory. Let's go right to the tutorial files

```
cd ~/lct/tutorial
```

Step 2 – Rebuilding The Code

Now we have to change the build process for the application so that it includes debugging information in the stab format, and the application is statically linked. Note that for the simple example we use here static linking doesn't really matter, but when you use ECT on a large application with multiple shared libraries, static linking for analysis allows ECT to check every function call made.

The Makefile looks like this (bring it up in your favorite editor)

```
CC=gcc
CFLAGS= -O2
LDFLAGS= -static
DEPS =
all: test1
test1: test1a.o test1b.o
        gcc -o $@ $^ $(CFLAGS) $(LDFLAGS)
%.o: %.c $(DEPS)
        $(CC) -c -o $@ $< $(CFLAGS)
.PHONY: clean
clean:
        rm -f *.o *~ core *.i *.s *.out
        rm -f test1
```

We'll start by changing the CFLAGS line from:

```
CFLAGS= -O2
```

to

```
CFLAGS= -O2 -gstabs+ -save-temps
```

Wherever you put gcc options in your Makefile, you will need to add "-gstabs+ -save-temps". This gives ECT the information needed to analyze the program and leaves behind the files you need to interpret the results.

Next we'll modify the linker flags to ensure the application is being statically linked. Change the LDFLAGS line:

```
LDFLAGS=
```

to

```
LDFLAGS= -static
```

The Makefile should now look like this:

```
CC=gcc
CFLAGS= -O2 -gstabs+ -save-temps
LDFLAGS= -static
all: test1
test1: test1a.o test1b.o
        gcc -o $@ $^ $(CFLAGS) $(LDFLAGS)
%.o: %.c $(DEPS)
        $(CC) -c -o $@ $< $(CFLAGS)
.PHONY: clean
clean:
        rm -f *.o *~ core *.i *.s *.out
        rm -f test1
```

Now you are almost ready to build this intensely complicated application. But first we have to verify that a few environment variables are set up. We have to assume that the GNU tools were set up in the default directory of /opt/sfw. You can quickly verify that the exports we are about to make will work by running the following commands:

```
pkgchk -l SFWgmake | grep gmake$
pkgchk -l SFWgcc33 | grep gcc$
pkgchk -l SUNWggrp | grep ggrep$
pkgchk -l SFWgcc33 | grep libstdc++.so
```

The output from each command will tell you the path to the GNU utilities like gmake, the path to the gcc compiler, and the path to the runtime libraries. If the results of these commands indicate your are installed somewhere other than /opt/sfw, you will have to modify the following commands appropriately to reflect the proper paths (The commands are ordered exactly as each entry here):

```
export PATH=/opt/sfw/gcc-3/bin:/opt/sfw/bin:/usr/sfw/bin/:$PATH
export LD_LIBRARY_PATH=/opt/sfw/gcc-3/lib/
```

At last you can build the application. Just run gmake from the command line and your application should build:

```
bash-2.05$ gmake
gcc -c -o test1a.o test1a.c -O2 -gstabs+ -save-temps
```

```
test1a.c: In function `main':
test1a.c:8: warning: return type of `main' is not `int'
gcc -c -o test1b.o test1b.c -O2 -gstabs+ -save-temps
gcc -o test1 test1a.o test1b.o -O2 -gstabs+ -save-temps -static
bash-2.05$
```

Step 3 – Running ECT

Now you finally get to run ECT against the code and see if there are any problems. Since we didn't bother including the ECT binaries in the PATH, we will run it as follows (we are still in ~/ect/tutorial) enter the following command:

```
../bin/ect.bash test1 test1.results
```

So you understand what is going on, ect.bash controls the analysis process, as it runs it calls various other programs for each stage of the analysis. To run ect.bash you pass it the executable you are analyzing and the name of a file to hold the results of the analysis.

Now that you've kicked off the analysis process you should see a bunch of output that looks like this:

```
0% Complete [11:43:10] - Staring ...
sparc-sun-solaris2.9
Solaris 9
This might take a long time to complete.. please do not interrupt it..
Using ...
      ECT Bin direct:  ../ect/bin
      executable file: test1
      results file:    test1.results
      database direct: ../ect/data/test1.DB
      objdump file:    ../ect/data/test1.DB/objdump.txt
10% Complete [11:43:25] - generated size table...
  [crtgdb]: generating function definitions
  [crtgdb]: generating gdb commands for function details.
  [crtgdb]: gdb version = 6
  [crtFuncDef]: generating type info

20% Complete [11:43:25] - generated function definitions table...
  [genObjFr]: generating function calls

30% Complete [11:43:27] - generated function references table...
```

```
40% Complete [11:43:27] - checked for risky API calls...
50% Complete [11:43:27] - checked for ioctl usage...
ERROR: not enough source information to build tFunctionRefParms.tbl
60% Complete [11:43:27] - checked for endian errors in function calls...
70% Complete [11:43:33] - checked for endian errors in global variable
declarations...
80% Complete [11:43:33] - checked for data size differences...
90% Complete [11:43:33] - checked for potentially invalid uses of
__BUILTIN...
100% Complete [11:43:33] - Formulated results ...

        Found 1 warnings and
              1 errors in
              2 files
```

You will notice partway through an error message came up "`ERROR: not enough source information to build tFunctionRefParms.tbl`". This is due to the simplicity of the tutorial application.

The end result is ECT found 1 error in the application and has 1 warning. You can view the report from the analysis by opening the file test1.report (there will always be an executable-name.report file at completion).

Step 4 – Analyzing The Results

The test.report for this run looks like this:

```
/export/home/joe-user/tut/test1b.i - Line: 220   E40002  Global Variable
"messedupvar" type mismatch "int" (Defined in "/export/home/joe-
user/tut/test1b.c" as "char *")
/opt/sfw/gcc-3/lib/gcc-lib/sparc-sun-solaris2.9/3.3.2/include/stdarg.h
- Line: 43       W60001  Potentially hazardous use of builtin:
__builtin_va_list
```

Now that we've run the tool and have our report, we need to review the results and see what remediations are necessary. We'll start with the error:

```
/export/home/joe-user/tut/test1b.i - Line: 220   E40002  Global Variable
"messedupvar" type mismatch "int" (Defined in "/export/home/joe-
user/tut/test1a.c" as "char *")
```

That line is telling us two important things, what kind of error we have and where we can find the offending code. First the error code E40002 tells us what we have to deal with, a type mismatch, a variable was defined differently in two files (All the error and warning messages are documented in the manual).

We can find the source of the problem on line 220 of test1b.i. That .i file is a temporary file created during compilation (has all include files included and macros expanded). Remember we added that -save-temps switch to the gcc options? That is so we can go into that file now and figure out the problem. If we look at line 220 of test1b.i, you can see that messedupvar is defined as an int.

```
218  # 2 "test1b.c" 2
219
220  extern int messedupvar;
```

The error message also tells us that when that variable was originally defined in test1a.c as a character pointer (line 3 in the following listing).

```
  1  #include <stdio.h>
  2
  3  char messedupvar[2] = "A";
  4
  5  void bad_function(void);
  6
  7  void main(void)
  8  {
  9          bad_function();
 10  }
```

The warnings we received alerts us to potential problems:

```
opt/sfw/gcc-3/lib/gcc-lib/sparc-sun-solaris2.9/3.3.2/include/stdarg.h -
Line: 43       W60001  Potentially hazardous use of builtin:
__builtin_va_list
```

In this case, since we aren't doing anything with variable argument functions, it is only an artifact from the standard include files and we shouldn't have to worry about changing anything. Each warning you receive should be investigated and not summarily dismissed.

In conclusion, we have taken you through the steps required to install ECT, run a code analysis, and interpret the results. The next step is to repeat the process on your application. The scale may be larger, the build processes may be more complicated, but the basic steps you need to take are exactly the same as you walked through here.