Annobin

The ANNOBIN plugin
(Annobin)
Version 12.0

Nick Clifton

This manual describes the ANNOBIN plugin and the `annocheck` program, and how you can use them to determine what security features were used when a program was built.

Table of Contents

1 What is Binary Annotation ?

Binary Annotation is a method for recording information about an application inside the application itself. It is an implementation of the `Watermark` specification defined here: `https://fedoraproject.org/wiki/Toolchain/Watermark`

Although mainly focused on recording security information, the system can be used to record any kind of data, even data not related to the application. One of the main goals of the system however is the ability to specify the address range over which a given piece of information is valid. So for example it is possible to specify that all of a program was compiled with the `-O2` option except for one special function which was compiled with `-O0` instead.

The range information is useful because it allows third parties to examine the binary and find out if its construction was consistent. IE that there are no gaps in the recorded information, and no special cases where a required feature was not active.

The system works by adding special sections to the application containing individual pieces of information along with an address range for which the information is valid. (Some effort has gone into the storing this information in a reasonably compact format).

The information is generated by a plugin that is attached to the compiler. The plugin extracts information from the internals of compiler and records them in the object file(s) being produced.

Note - the plugin method is just one way of generating the information. Any interested party can create and add information to the object file, providing that they follow the Watermark specification.

The information can be extracted from files via the use of tools like `readelf` and `objdump`. The `annobin` package itself includes a program called `annocheck` which can can also examine this information. Details on this program can be found elsewhere in this documentation.

Experience has shown however that storing the range information along with the data does tend to significantly increase the size of programs. So the system also provides an alternative implementation which uses a more compact format, at the cose of dropping the range data.

2 How to add Binary Annotations to your application.

Normally the option to enable the recording of binary annotation notes is enabled automatically by the build system, so no user intervention is required. On Fedora and RHEL based systems this is handled by the '`redhat-rpm-config`' package.

Currently the binary annotations are generated by a plugin to the compiler (`GCC`, `clang` or `llvm`). This does mean that files that are not compiled by any of these compilers will not gain any annotations, although there is an optional assembler switch to add some basic notes if none are present in the input files.

If the build system being used does not automatically enable the `annobin` plugin then it can be specifically added to the compiler command line by adding the `-fplugin=annobin` (for gcc) or `-fplugin=annobin-for-clang` (for clang) or `-fplugin=annobin-for-llvm` (for LLVM) option. It may also be necessary to tell the compiler where to find the plugin by adding the `-iplugindir=` option, although this should only be necessary if the plugin is installed in an unusual place.

If it is desired to disable the recording of binary annotations then the `-fplugin-arg-annobin-disable` (for `gcc`) or `-Xclang -plugin-arg-annobin-disable` (for `clang` or `llvm`) can be used. Note - these options must be placed *after* the `-fplugin=annobin` option.

On Fedora and RHEL systems the plugin can be disabled entirely for all compilations in a package by adding `%undefine _annotated_build` to the spec file.

The plugin accepts a small selection of command line arguments, all accessed by passing `-fplugin-arg-annobin-<option>` (for `gcc`) or `-Xclang -plugin-arg-annobin-<option>` (for `clang` or `llvm`) on the command line. These options must be placed on the command line after the plugin itself is mentioned. Note - not all versions of the plugin accept all of these options.

In addition it is possible to pass options via the `ANNOBIN` environment variable. Multiple arguments must be separated by commas, and arguments that need a value must use an equals sign rather than a space or colon.

Also - in order to support backwards compatibility - the LLVM plugin detects the `ANNOBIN_VERBOSE` enviroment variable and turns on verbose mode if it is present.

The supported options are:

`disable`
`enable`
: Either disable or enable the plugin. The default is for the plugin to be enabled.

`help`
: Display a list of supported options on the standard output. This is in addition to whatever else the plugin has been instructed to do.

`version` Display the version of the plugin on the standard output. This is in addition to whatever else the plugin has been instructed to do.

`verbose` Report the actions that the plugin is taking. If invoked for a second time on the command line the plugin will be very verbose.

`function-verbose`
: Report the generation of function specific notes. This indicates that the named function was compiled with different options from those that were globally enabled.

`stack-size-notes`
`no-stack-size-notes`
: Do, or do not, record information about the stack requirements of functions in the executable. This feature is disabled by default as these notes can take up a lot of extra room if the executable contains a lot of functions.

`stack-threshold=`*`N`*
: If stack size requirements are being recorded then this option sets the minimum value to record. Functions which require less than `N` bytes of static stack space will not have their requirements recorded. If not set, then `N` defaults to 1024.

`global-file-syms`
`no-global-file-syms`
: If enabled the `global-file-syms` option will create globally visible, unique symbols to mark the start and end of the compiled code. This can be desirable if a program consists of multiple source files with the same name, or if it links to a library that was built with source files of the same name as the program itself. The disadvantage of this feature however is that the unique names are based upon the time of the build, so repeated builds of the same source will have different symbol names inside it. This breaks the functionality of the build-id system which is meant to identify similar builds created at different times. This feature is disabled by default, and if enabled can be disabled again via the `no-global-file-syms` option.

`attach`
`no-attach`
: When gcc compiles code with the `-ffunction-sections` option active it will place each function into its own section. When the `annobin attach` option is active the plugin will attempt to attach the function section to a group containing the notes and relocations for the function. In that way, if the linker decides to discard the function, it will also know that it should discard the notes and relocations as well.

The default is `attach`, but this can be disabled via the `no-attach` option. Note however that if both `attach` and `link-order` are disabled then note generation for function sections will not work properly.

`link-order`
`no-link-order`
: As an alternative to using section groups and a special assembler directive the plugin can use a feature of the ELF `SHF_LINK_ORDER` flag which tells the linker that it should discard a section if the section it is linked to is also being discarded. This behaviour is enabled by the `link-order` option.

`rename`
: Adds an extra prefix to the symbol names generated by the `annobin` plugin. This allows the plugin to be run twice on the same executable, which can be useful for debugging and build testing.

`active-checks`
`no-active-checks`
: The `annobin` plugin will normally generate warning messages if it detects that certain preprocessor command line options are missing or misspelt. The `active-checks` option changes the warnings into errors, just as if `-Werror` had been specified. The `no-active-checks` option disables the messages entirely.

Currently the plugin checks for these issues:

`Missing FORTIFY_SOURCE`
: This warning is generated when neither `-D_FORTIFY_SOURCE=2` nor `-D_FORTIFY_SOURCE=3` have been provided on the command line and the `-flto` option has been enabled.

Nomrally this problem would be detected by the `annocheck` tool, but LTO compilation hides preprocessor options, so information about them cannot be passed on by the plugin. This is why the plugin will generate a warning message when the `_FORTIFY_SOURCE` option is missing and LTO is enabled.

`-D_FORTIFY_SOURCE typo`
: The plugin will warn if the `-D_FORTIFY_SOURCE` option is spelt as either `-DFORTIFY_SOURCE` or `-D__FORTIFY_SOURCE`.

`-D_GLIBCXX_ASSERTIONS typo`
: The plugin will warn if the `-D_GLIBCXX_ASSERTIONS` option is spelt as either `-DGLIBCXX_ASSERTIONS` or `-D__GLIBCXX_ASSERTIONS`.

Note - in the future the `annobin` plugin might be extended to produce warning messages for other missing command line options.

Note - as a workaround for certain tests generated by the `autoconf` tool the warning message will not be produced if the input source filename starts with `conftest.`. In these cases autoconf is usually checking to see if a warning will be produced for some other reason, and so the `annobin` warning would get in the way. If the `active-checks` option has been enabled however, an error message will still be generated.

`dynamic-notes`
`no-dynamic-notes`
`static-notes`
`no-static-notes`

These options are deprecated.

`ppc64-nops`
`no-ppc64-nops`

This option either enables or disables the insertion of NOP instructions in the some of the code sections of PowerPC64 binaries. This is necessary to avoid problems with the `elflint` program which will complain about binaries built without this option enabled. The option is enabled by default, but since it does increase the size of compiled programs by a small amount, the `no-ppc64-nops` is provided in order to turn it off.

`note-format=note|string`

This option chooses the format used to store the information generated by the plugin. The possibilities are:

`note`	Store the information as ELF format notes in the `.gnu.build.attributes` section.
`string`	Store the information as mergeable strings in the `.annobin.notes` section.

The default is `note`.

The plugins record information appropriate to the compiler that is running them. So the `gcc` plugin records information about the following options:

`-D_FORTIFY_SOURCE=[2|3]`
`-D_GLIBCXX_ASSERTIONS`
`-O`

`-Wall`

`-fPIC`

`-fPIE`

`-fcf-protection`
`-finstrument_functions`
`-flto`

`-fomit-frame-pointer`
`-fprofile`
`-fprofile-arcs`
`-fsanitize`
`-fshort-enums`
`-fstack-clash-protection`
`-fstack-protector`
`-g`

`-mbranch-protection (AArch64)`
`-mstack-realign (i386)`
`-mtls-size (PowerPC)`

The `Clang` plugin records information on the following command line options:

`-O`

`-Wall`

`-fPIC`

`-fPIE`

`-fcf-protection-branch`
`-fcf-protection-return`
`-fsanitize=cfi-cast-strict`
`-fsanitize=safe-stack`
`-fspeculative-load-hardening`
`-fstack-protector-strong`

Note - if LTO compilation is enabled (-flto) then any data recorded by the Clang plugin is ignored when the object file is recompiled by the LLVM backend. Hence when using LTO and Clang it is best to enable the LLVM plugin.

The `LLVM` plugin records information on the following command line options:

`-D_FORTIFY_SOURCE=[2|3]`
`-O`

`-Wall`

`-flto`

`-fPIC`

`-fPIE`

```
-fcf-protection-branch
-fcf-protection-return
-fsanitize=safe-stack
-fstack-protector-strong
-g
```

3 How to examine the information stored in the binary.

The information is stored in a binary in either the ELF Note format inside a special section called `.gnu.build.attributes`, or else as ordinary strings inside a section called `.annobin.notes`.

The `readelf` program from the `binutils` package can extract and display these notes. (Adding the `--wide` option is also helpful).

If the information is held in the ELF note format then readelf's `--notes` option will display them. Here is an example of the output:

```
Displaying notes found in: .gnu.build.attributes
  Owner                          Data size Description
  GA$<version>3p3                0x00000010 OPEN     Applies to region from 0x8a0 to 0x8c
  GA$<tool>gcc 7.2.1 20170915    0x00000000 OPEN     Applies to region from 0x8a0 to 0x8c
  GA*GOW:0x452b                  0x00000000 OPEN     Applies to region from 0x8a0 to 0x8c
  GA*<stack prot>strong          0x00000000 OPEN     Applies to region from 0x8a0 to 0x8c
  GA*GOW:0x412b                  0x00000010 func     Applies to region from 0x8c0 to 0x8c
```

This shows various different pieces of information, including the fact that the notes were produced using version 3 of the specification, and version 3 of the plugin. The binary was built by gcc version 7.2.1 and the -fstack-protector-strong option was enabled on the command line. The program was compiled with -O2 enabled except the baz() function which was compiled with -O0 instead.

The most complicated part of the notes is the owner field. This is used to encode the type of note as well as its value and possibly extra data as well. The format of the field is explained in detail in the Watermark specification, but it basically consists of the letters '`G`' and '`A`' followed by an encoding character (one of '`*$!+`') and then a type character and finally the value.

The notes are always four byte aligned, even on 64-bit systems. This does mean that consumers of the notes may have to read 8-byte wide values from 4-byte aligned addresses, and that producers of the notes may have to generate unaligned relocs when creating them.

If the information is held as strings then readelf's `-p.annobin.notes` option will display them. Here is an example of the output:

```
String dump of section '.annobin.notes':
  [     0]  AV:4.p.1200
  [     c]  RV:running gcc 12.2.1 20221121
  [    2b]  BV:annobin gcc 12.2.1 20221121
  [    4a]  PN:annobin
  [    55]  GW:0x290540
  [    61]  SP:3
```

Most of the notes have a reasonably self explanatory name and value. The exception are the `version` and `GOW` notes, which are included in the table below.

3.1 Encoding Protocol and Producer Versions

The `version` note encodes the version of the Watermark specification used and the version of the tool used to generate the notes. Typically the protocol version will be 3 or 4 and the plugin version will be 11 or 12. It also encodes the tool used to generate the notes as a single character. The following characters are used:

`L`	The notes have been produced by the Clang plugin.
`V`	The notes have been produced by the LLVM plugin.
`a`	The notes have been produced by the assembler.
`c`	The notes have been produced by the gcc plugin for the .text.cold section.
`e`	The notes have been produced by the gcc plugin for the .text.exit section.
`g`	The notes have been produced by the gcc plugin when running in LTO mode.
`h`	The notes have been produced by the gcc plugin for the .text.hot section.
`l`	The notes have been produced by the linker.
`p`	The notes have been produced by the gcc plugin.
`s`	The notes have been produced by the gcc plugin for the .text.startup section.

3.2 Encoding Stack Protections

The stack protection note (value 2) encodes the setting of the `-fstack-protector` option. Possible values are:

`0`	Not compiled with any setting of `-fstack-protector` (or the setting is unknown).
`1`	Compiled with just `-fstack-protector`.
`2`	Compiled with `-fstack-protector-all`.
`3`	Compiled with `-fstack-protector-strong`.
`4`	Compiled with `-fstack-protector-explicit`.

3.3 Encoding Position Independence

The `Position Independence Status` note encodes the setting of the `-fpic`/`-fpie` used when compiling the program. The value of the note can be

`0` Static code, ie neither pic nor pie.

`1` Compiled with `-fpic`.

`2` Compiled with `-fPIC`.

`3` Compiled with `-fpie`.

`4` Compiled with `-fPIE`

If both `pic` and `pie` have been specified on the command line then `pie` takes the precedence in the encoding.

3.4 Encoding Optimization and Debugging Levels

The `GOW` note encodes the optimization level (`-O`) and debugging level (`-g`) used when compiling a binary. In order to save space this is stored as a bit field with the bits having the following meanings:

`bits 0 - 2`
: The debug type, ie DBX, DWARF, VMS or XCOFF. As specified by the `-gstabs`, `-gdwarf`, `-gvms` and `-gxcoff` options.

`bit 3` Set if GNU extensions to the debug type have been enabled.

`bits 4 - 5`
: The debug info level ie TERSE, NORMAL or VERBOSE as set by the `-g<level>` option.

`bits 6 - 8`
: The DWARF version, if DWARF is being generated. Set by the `-gdwarf-<version>` option.

`bits 9 - 10`
: The optimization level as set by the `-O<number>` option. Levels above 3 are treated as if they were 3.

`bit 11` Set if the optimize-for-size option (`-Os`) is enabled.

`bit 12` Set if the inaccurate-but-fast optimization option (`-Ofast`) has been enabled.

`bit 13` Set if the optimize-with-debugging option (`-Og`) has been enabled.

`bit 14` Set if the enable most warnings option (`-Wall`) has been enabled.

`bit 15` Set if the format security warning option (`-Wformat-security`) has been enabled.

`bit 16` Set if LTO compilation has been enabled.

`bit 17` Set if LTO compilation has not been enabled.

This bit is here so that tools can detect notes created by earlier versions of annobin which did not set any bits higher than 15.

`bits 18 - 19`
: These bits are used to record the setting of gcc's `-ftrivial-auto-var-init` command line option. If both bits are clear then the option is not supported by the compiler. If bit 18 is set but bit 19 is clear, then the compiler supports the option, but either it was not used, or it was used but the option's value was `skip`. Otherwise if bit 19 is clear then the option was used but its value was set to `pattern` (which is inappropriate for production binaries) or if bit 19 is set then the option was used and its value was set to `zero`.

`bits 20 - 21`
: These bits are used to record the setting of gcc's `-fzero-call-used-regs` command line option. If both bits are clear the option is not supported by the compiler. Otherwise if bit 20 is set (and bit 21 is clear) then the option is supported but it was not used, or it was used, but its value was set to `skip`. Otherwise both bits 20 and 21 should be set, indicating that the option was used and a value other than `skip` was used.

`bits 22 - 23`
: These bits are used to record the setting of gcc's `-Wimplicit-int` warning. Enabling this warning is important as variables with an implicit type of int can cause problems when they are used in situations where they are expected to be compatible with pointers. The meanings of the two bits are as follows:

  `bit22 = 0, bit23 = 0`
  : The annobin plugin did not record any information about this warning, probably because the plugin is an old version.

  `bit22 = 1, bit23 = 0`
  : The warning has been disabled.

  `bit22 = 0, bit23 = 1`
  : The warning has its default value. It has not been altered via a command line option.

  `bit22 = 1, bit23 = 1`
  : The warning has been enabled via a command line option.

  Note - whilst this warning is normally enabled it can be manually disabled. Hence the need for this extra test.

  Note - this test is only really relevent to C source code. Other languages are not affected.

`bits 24 - 25`
: These bits are used to record the setting of gcc's `-Wimplicit-function-declaration` warning. Enabling this warning is im-

portant as functions that are assumed to accept int parameters or return int values can cause problems when pointers are involved. The meanings of the two bits are as follows:

`bit22 = 0, bit23 = 0`
: The annobin plugin did not record any information about this warning, probably because the plugin is an old version.

`bit22 = 1, bit23 = 0`
: The warning has been disabled.

`bit22 = 0, bit23 = 1`
: The warning has its default value. It has not been altered via a command line option.

`bit22 = 1, bit23 = 1`
: The warning has been enabled via a command line option.

Note - whilst this warning is normally enabled it can be manually disabled. Hence the need for this extra test.

Note - this test is only really relevent to C source code. Other languages are not affected.

`bits 26 - 28`
: These bits record the settings of gcc's flexible array strengthening. Bit 26 is set only if the feature is supported by the compiler. If bit 26 is not set then bits 27 to 28 should be ignored. Bit 27 is set if the `-Wstrict-flex-arrays` warnings is enabled. Bit 28 is set if the `-fstrict-flex-arrays` option has been set to a value greater than zero.

The other bits are not currently used and should be set to zero so they can be used in future extensions to the specification.

3.5 Encoding Control Flow Protection

Records the setting of the `-cf-protection` option. This is a bit mask using the following bits, based upon the definition of the `enum cf_protection_level` from gcc's `flag-types.h` header file:

`bit 0` Branches are protected. (ie `-fcf-protection=branch`).

`bit 1` Returns are protected. (ie `-fcf-protection=return`).

`bit 2` If set, this indicates that the other bits were explicitly set by an option on the gcc command line. Otherwise those bits were implicitly set by either other options or the backend concerned.

If both bits 0 and 1 are set then this implies the `-fcf-protection=full` option, and if neither are set then this implies the `-fcf-protection=none` option.

Note - in order to avoid storing a value of 0 in the note (which can be confused with a NUL-byte to indicate the end of a string), the value stored is biased by 1.

3.6 Encoding the Size of Enumerations

Record the value of the `-fshort-enums` option. Possible values are:

`true` The `-fshort-enums` option has been enabled.

`false` The `-fshort-enums` option has not been enabled.

3.7 Encoding Instrumentation Options

Records the enablement of various code instrumentation options. Note - this note is only produced if one or more of these options are enabled.

The note encodes four values, separate by the forward slash (/) character. These values are:

`sanitization`
: Enabled via a plethora of `-fsanitize=...` options these tell gcc to add extra code to help with various different types of error checking features.

`function instrumentation`
: Enabled via gcc's `-finstrument-functions` option, this adds special function calls at the entry and exit point of every normal function.

`profiling`
: Enabled via gcc's `-p` or `-pg` options, this adds instrumentation to the compiled code that generates output suitable for analysis via the `prof` or `gprof` programs.

`arc profiling`
: Enabled via gcc's `-fprofile-arc` option, or one of the meta-profiling options, this option adds code to record how many times every branch and function call is executed.

3.8 Encoding Notes in a string format

If the notes are being recorded in a string format, via the `note-formt=string` command line options, then the strings take the form of a two letter prefix, followed by a colon and then the value associated with the note. If the note is likely to trigger a failure result in annocheck then the filename of the source file will follow, separated by a space. If the note is specific to a single function then the function name will follow on afterwards, again separated by a space.

The currently accepted two letter prefixes are:

`AV`	Records the version of the plugin.
`BV`	Records the version of the compiler that built the plugin.
`CF`	Records the setting of any control flow security options.
`FL`	Records the level of the `-D_FORTIFY_LEVEL` option.
`FP`	Records the use of the frame pointer.
`GA`	Records the setting of the `-D_GLIBCCXX_ASSERTIONS` command line option.
`GW`	Records the GOW values.
`PF`	Records any profiling options.
`PI`	Records any PIC settings.
`PN`	Records the name of the plugin.
`RV`	Records the version of the compiler that is running the plugin.
`SC`	Records the setting of any stack clash protection.
`SE`	Records the use of short enums.
`SP`	Records the use of an stack protector options.
`aA`	Records the ABI selected. (AArch64 only).
`aB`	Records the use of Branch Target Identification. (Aarch64 only).
`iS`	Records the use of the stack realign option. (x86 only).
`pA`	Records the ABI selected. (PowerPC only).
`xA`	Records the ABI selected. (x86_64 only).

Each value represents a setting of an internal gcc flag variable. The exact meaning of the values is specific to gcc, but any non-zero number means that the feature has been enabled in some way.

4 Analysing binary files.

```
annocheck
  [-h | –help]
  [–help-tool]
  [–version]
  [-v | –verbose]
  [-q | –quiet]
  [-i | –ignore-unknown]
  [-r | –report-unknown]
  [-f | –follow-links]
  [-I | –ignore-links]
  [–debug-rpm=file]
  [–dwarf-dir=dir]
  [-p text | –prefix=text]
  [-t dir | –tmpdir=dir]
  [-u | –use-debuginfod]
  [-n | –no-use-debuginfod]
  [–enable-tool]
  [–disable-tool]
  [–tool]
  [–tool-option]
  file...
```

The `annocheck` program can analyse binary files and report information about them. It is designed to be modular, with a set of self-contained tools providing the checking functionality. Currently the following tools are implemented:

The `annocheck` program is able to scan inside rpm files and libraries. It will automatically recurse into any directories that are specified on the command line. In addition `annocheck` knows how to find debug information held in separate debug files, and it will search for these whenever it needs the resources that they contain.

New tools can be added to the annocheck framework by creating a new source file and including it in the `Makefile` used to build `annocheck`. The modular nature of `annocheck` means that nothing else needs to be updated.

New tools must fill out a `struct checker` structure (defined in `annocheck.h`) and they must define a constructor function that calls `annocheck_add_checker` to register their presence at program start-up.

The `annocheck` program supports some generic command line options that are used regardless of which tools are enabled.

`--debug-rpm=`*`file`*
: Look in *file* for separate dwarf debug information.

`--dwarf-dir=`*`dir`*
: Look in *dir* for separate dwarf debug information files.

`--help`
`-h`
: Displays the generic annobin usage information and then exits.

`--help-`*`tool`*
: Display the usage information for *tool* and then exits.

`--report-unknown`
`--ignore-unknown`
`-r`
`-i`
: If enabled, unknown file types are reported when they are encountered. This includes non-ELF format files, block devices and so on. Directories are not considered to be unknown and are automatically descended.

  The default setting depends upon the file being processed. For rpm files the default is to ignore unknowns, since these often contain non-executable files. For other file types, including directories, the default is to report unknown files.

`--ignore-links`
`--follow-links`
`-I`
`-f`
: Specifies whether symbolic links should be followed or ignored.

  The default setting depends upon the file being processed. For rpm files the default is to ignore symbolic links, since these often unresolveable. For other file types, including directories, the default is to follow the links.

`--prefix=`*`text`*
`-p` *`text`*
: Include *text* in the output description.

`--quiet`
`-q`
: Do not print anything, just return an exit status.

`--tmpdir=`*`dir`*
`-t` *`dir`*
: Use *dir* as a directory for holding temporary files.

`--verbose`
`-v`
: Produce informational messages whilst working. Repeat for more information.

`--version`
: Report the version of the tool and then exit.

`--use-debuginfod`
`-u`
: Enable the use of the debuginfod service to download debuginfo rpms. This feature is enabled by default, but it is only active if support for the debuginfod server has been compiled in to annocheck.

`--no-use-debuginfod`
`-n`
: Do not use the debuginfod service, even if it is available.

`--enable-`*`tool`*
: Enable *tool*. Most tools are disabled by default and so need to be enabled via this option before they will act.

`--disable-tool`
: Disable *tool*. Normally used to disable the hardening checker, which is enabled by default.

`--tool`
: Enable *tool* and disable all other tools.

`--tool-option`
: Pass *option* on to *tool*.

Any other command line options will be passed to the tools in turn in order to give them a chance to claim and process them.

4.1 The builder checker.

```
annocheck
  –enable-builtby
  [–all]
  [–tool=name]
  [–nottool=name]
  [–no-version-info]
  [–no-lang-info]
  file...
```

The *built-by* tool is disabled by default, but it can be enabled by the command line option `--enable-builtby`. The tool checks the specified files to see if any information is stored about how the file was built and the source languages involved.

Since the hardening checker is enabled by default it may also be useful to add the `--disable-hardened` option to the command line.

The tool supports a few command line options to customise its behaviour:

`--all`
: Report all builder identification strings. The tool has several different heuristics for determining the builder. By default it will report the information return by the first successful heuristic. If the `--all` option is enabled then all successful results will be returned.

`--tool=name`
: This option can be used to restrict the output to only those files which were built by a specific tool. This can be useful when scanning a directory full of files searching for those built by a particular compiler. This option can be used multiple times in order to allow a selection of builders to be reported.

`--nottool=NAME`
: This option can be used to restrict the output to only those files which were not built by a specific tool. This can be useful when scanning a directory full of files searching for those that were not built by a particular compiler. This option can be used multiple times in order to allow multiple builders to be hidden.

`--lang=`*`name`*
: This option can be used to restrict the output to only those files which were written in a specific high level language. Note - not all binaries include information about the source code language(s), so this option may not be completely effective. This option can be used multiple times in order to allow a broader selection of languages to be reported.

`--notlang=`*`NAME`*
: This option can be used to restrict the output to only those files which were not written in a specific high level language. Note - not all binaries include information about the source code language(s), so this option may not be completely effective. This option can be used multiple times in order to allow a broader selection of languages to be hidden.

`--no-version-info`
`--lang-info`
: By default `built-by` will report the version information for the builders that it detects. Enabling the `--no-version-info` option will prevent this information from being displayed.

  If necessary the feature can be re-enabled by the `--version-info` option.

`--no-lang-info`
`--lang-info`
: By default `built-by` will report the high level language(s) of the sources used to build the program - if they have been recorded. Enabling the `--no-lang-info` option will prevent this information from being displayed.

  If necessary the feature can be re-enabled by the `--lang-info` option.

`--no-tool-info`
`--tool-info`
: By default `built-by` will report the tool(s) used to build the program. Enabling the `--no-build-info` option will prevent this information from being displayed, meaning that only the high level language information will shown. Enabling this option and the `--no-lang-info` option effectively renders `built-by` redundant.

  If necessary the feature can be re-enabled by the `--tool-info` option.

4.2 The Hardened security checker.

```
annocheck
```

[–**skip-***name*[=*funcname*]]
[–**test-***name*]
[–**skip-all**]
[–**test-all**]
[–**skip-future**]
[–**test-future**]
[–**test-unicode-all**]
[–**test-unicode-suspicious**]
[–**profile**=*release*]
[–**ignore-gaps**]
[–**report-gaps**]
[–**fixed-format-messages**]
[–**disable-colour**]
[–**enable-colour**]
[–**disable-hardened**]
[–**enable-hardened**]
[–**full-filenames**]
[–**base-filenames**]
[–**suppress-version-warnings**]
[–**no-urls**]
[–**provide-urls**]
file...

The *hardened* tool checks that the specified files were built with specific security hardening features enabled. The features that are tested can be controlled via command line options, but the default is to test for all of them.

The tool was originally built to assist in the implementation of security features for Fedora, although it does now check for more things than are described in that document: `https://fedoraproject.org/wiki/Security_Features`

New tests can be added to the *hardened* checker by adding an entry in the *tests* array defined in `hardened.c` and then creating the necessary code to support the test. There is more information on this process in this blog: `https://developers.redhat.com/articles/2021/07/15/build-your-own-tool-search-code-sequences-binary-files`

Currently the *hardened* tool can run the following tests. Each test listed here starts with a short section describing the reason for the test, a probable solution to fix the test, criteria for when the test can be ignored and some examples of the error messages that are produced by annocheck when the test goes wrong.

4.2.1 The Tests Run By Annocheck

4.2.1.1 The auto-var-init test

```
  Problem:  An attacker could extract information from uninitialised lo-
cations on the stack
  Fix By:   Add -fauto-var-init=zero to the compiler command line
  Waive If: The overhead of initializing auto variables is too high
```

```
Example:  FAIL: auto-var-init test because -ftrivial-auto-var-init not used or set to
```

This is a future test. It is not enabled by default. It checks a security feature that may not be widely available or enforced.

This test checks to make sure that programs have been compiled with the `-ftrivial-auto-var-init=zero` command line option. This option ensures that all local variables are initialised, even if they are not used. This includes any padding between structure fields or unused array entries. If this is not done then a potential attacker might be able to access the unitialized memory and extract information.

The test can be enabled via the `--test-auto-var-init` option and disabled by the `--skip-auto-var-init` option. It is also enabled if the `--test-future` option is specified and disabled if the `skip-future` option is specified.

4.2.1.2 The bind-now test

```
Problem:  An attacker could intercept calls to shared library functions
Fix By:   Add -Wl,-z,now to final link command line
Waive If: No shared libraries used

Example:  FAIL: bind-now test because not linked with -Wl,-z,now
```

This test checks that lazy binding is not enabled in the binary. Lazy binding can be used to delay resolving the links between an application and any shared libraries that it uses:

`https://www.airs.com/blog/archives/41`

Using lazy binding provides a faster start-up for an application since this resolving process is not performed until a function call is made to a specific library. But it is also a security vulnerability since an attacker could replace the binding with a link to their own code. Hence for security purposes immediate binding rather than lazy binding should be used.

The type of binding is selected via a linker command line option, and on a compiler command line the secure version usually looks like `-Wl,-z,now`. The lazy binding option is `-Wl,-z,lazy` although some linkers are configured to use lazy binding by default, in which case just the absence of the `-Wl,-z,now` option is enough to trigger this test.

Whilst important, this test can be ignored if the binary does not use any shared libraries.

Note - this test is automatically disabled if the `--profile=el7` option is used.

The test can be disabled via the `--skip-bind-now` option and re-enabled by the `--test-bind-now` option.

4.2.1.3 The branch-protection test

```
Problem:  Unprotected AArch64 binaries are vulnerable to ROP/JOP style attacks
```

```
  Fix By:   Compile with -mbranch-protection=standard
  Waive If: Not running on AArch64
  Waive If: The application will not run on Fedora 35 or later.
  Waive If: The application will not run on newer AArch64 cores.

  Example:  FAIL: branch protection test because not enabled
  Example:  FAIL: branch protection test because only partially enabled
  Example:  FAIL: branch protection test because .note.gnu.property sec-
tion not found (it is needed for branch protection support)
  Example:  FAIL: branch protection test because the -mbranch-protection op-
tion was not used
```

AArch64 processors are vulnerable to a class of attack known as *ROP* and *JOP* style attacks. Preventing this kind of exploit requires assistance from the hardware itself, in the form of new instructions that need to be inserted by the compiler, and new bits in the core's status that need to be set.

This test checks to see if the compile time option to enable the security feature was used. There are four levels of security available, ranging from none through partial (some functions are protected others are not) to full. The test checks that full security has been enabled.

The security feature is enabled by compiling with the `-mbranch-protection=standard` gcc command line option.

Note - these security features are only found on newer versions of the AArch64 architecture, and they need a compiler and a loader that will support them. Currently this means Fedora 35 or later, but not RHEL-8 or RHEL-9.

If an assembler source file is used as part of an application then it too needs to be updated. Any location in the source code where an indirect branch or function call can land must now have *BTI* as the first instruction executed. This instruction is safe to use even in code that will not be executed in a BTI-enabled environment as it translates into a no-op instruction if not needed.

In addition the assembler needs a note to indicate that it now supports BTI. This note can be added via including this code snippet in the sources:

```
.pushsection .note.gnu.property, "a"
.align 3
.word 2f - 1f
.word 4f - 3f
.word 5            /* NT_GNU_PROPERTY_TYPE_0 */
1:  .asciz "GNU"

2:  .align 3
3:  .word 0xc0000000   /* type: GNU_PROPERTY_AARCH64_FEATURE_1_AND */
.word 6f - 5f      /* size */
5:  .word 1            /* value: GNU_PROPERTY_AARCH64_FEATURE_1_BTI */

6:  .align 3
4:  .popsection
```

Note - this test is the inverse of the Section 4.2.1.20 [Test not branch protection], page 33, test and directly related to the Section 4.2.1.6 [Test dynamic tags], page 26, test.

Note - this test is automatically enabled if one of the following profile options is used:

```
--profile=rawhide
--profile=f38
--profile=f37
--profile=f36
--profile=el10
```

The test is automatically disabled if one of the other profile options is used, ie:

```
--profile=el7
--profile=el8
--profile=el9
--profile=f35
```

If necessary the test can be disabled via the `--skip-branch-protection` option and re-enabled via the `--test-branch-protection` option.

4.2.1.4 The cf-protection test

```
    Problem:  An attacker could compromise an unprotected binary
    Fix By:   Compiling with -fcf-protection=full
    Waive If: The application will not run on the latest Intel hardware
    Waive If: The application is built by a compiler that does not support CET

    Example:  FAIL: cf-protection test because only branch protection enabled
    Example:  FAIL: cf-protection test because only return protection enabled
    Example:  FAIL: cf-protection test because no protection enabled
    Example:  FAIL: cf-protection test because insufficient Control Flow sanitization
    Example:  FAIL: cf-protection test because no .note.gnu.property section = no con-
  trol flow information
    Example:  FAIL: cf-protection test because CET enabling note missing
    Example:  FAIL: cf-protection test because control flow protection is not enabled
```

Intel have introduced a new security feature called *CET* to their Tiger Lake and newer cores:

`https://www.intel.com/content/www/us/en/newsroom/opinion/intel-cet-answers-call-protect-common-malware-threats.html`

This test checks to see that this feature is enabled. Normally this is done by compiling the code with the `-fcf-protection` or `-fcf-protection=full` command line option enabled. (The first form of the option is an alias for the second, but the second form is preferred as it explicitly shows that all of the control flow protection features are being enabled).

But if an application contains assembler code, or it is linked against a library that has not been built with the protection enabled, or it is built by a compiler that does not support *CET* then this test can fail.

The feature has to be enabled in the compiler as it involves inserting new instructions into the compiled code. The feature is also an all-or-nothing type proposition for any process. Either all of the code in the process must have been built to support CET - in which case the feature can be enabled - or if even a single component does not support CET then it must be disabled for the entire process.

In order to enforce this the compiler inserts a special note into compiled object files (the .note.gnu.property section referred to above). The note indicates that CET is supported, as well as details of the minimum x86 architecture revision needed and so on.

Then when the object files are linked together to create the executable the linker checks all of these notes, and if any object file or library is missing one then it does not put a note in the output executable. Alternatively if all of the object files (and libraries of course) do have notes, but one or more of them do not have the CET-is-enabled flag, then the linker copies the notes into the executable, but always clears the CET-is-enabled flag.

Finally when a program is executed the run-time loader checks this note and if the CET-is-enabled flag is present then it enables the CET feature in the hardware.

Fixing this check either means enabling the `-fcf-protection=full` (for gcc) or the `-fcf-protection-branch` and `-fcf-protection-return` options (for Clang).

If an assembler source file is used as part of an application then it too needs to be updated. Any location in the source code where an indirect branch or function call can land must now have either *ENDBR64* (for 64-bit assembler) or *ENDBR32* (for 32-bit assembler) as the first instruction executed.

In addition the assembler needs a note to indicate that it now supports CET. This note can be added via including this code snippet in the sources:

```
.section .note.gnu.property,"a"
.align 8
.long  1f - 0f
.long  4f - 1f
.long  5
0:
.string  "GNU"
1:
.align 8
.long  0xc0000002
.long  3f - 2f
2:
.long  0x3
3:
```

```
.align 8
4:
```

If necessary the test can be disabled via the `--skip-cf-protection` option and re-enabled via the `--test-cf-protection` option.

For more information on CET see: `https: / / www . intel . com / content / dam / develop / external / us / en / documents / catc17-introduction-intel-cet-844137.pdf`

4.2.1.5 The dynamic-segment test

```
Problem:  Programs with more than one dynamic section will not be loaded properly
Fix By:   Fix assembler source code and/or linker script
Waive If: Don't.

Example:  FAIL: dynamic segment test because multiple dynamic sections detected
```

Dynamic executables must have a dynamic section which contains information that is used by the loader at program startup. The loader however only expects there to be one dynamic section in a program, and it does not cope it there are more than one. Normally this is not an issue however as the linker will ensure that there is only one dynamic section. It is possible however to use a custom linker script to create more than one dynamic section, or to write some assembler code specifically designed to create multiple dynamic sections.

If necessary the test can be disabled via the `--skip-dynamic-segment` option and re-enabled via the `--test-dynamic-segment` option.

4.2.1.6 The dynamic-tags test

```
Problem:  Unprotected AArch64 binaries are vulnerable to ROP/JOP style attacks
Fix By:   Compile with -mbranch-protection=standard
Waive If: Not running on AArch64
Waive If: The application will not run on Fedora 35 or later.
Waive If: The application will not run on newer AArch64 cores.

Example:  FAIL: dynamic tags test because BTI_PLT and PAC_PLT flags miss-
ing from the dynamic tags
Example:  FAIL: dynamic tags test because BTI_PLT flag is missing from the dy-
namic tags
Example:  FAIL: dynamic tags test because PAC_PLT flag is missing from the dy-
namic tags
Example:  FAIL: dynamic tags test because no dynamic tags found
```

AArch64 processors are vulnerable to a class of attack known as *ROP* and *JOP* style attacks. Preventing this kind of exploit requires assistance from the hardware itself, in the form of new instructions that need to be inserted by the compiler, and new bits in the core's status that need to be set.

This test checks to see if executable binaries have been marked as supporting the necessary security features to prevent this kind of attack. (The BTI_PLT and PAC_PLT flags mentioned in the failure messages). If they are marked then the runtime loader can enable the features in the processor core. This marking is done by setting flags in the tags found in the dynamic section of the executable. If the flags are missing then the executable is considered to be unprotected.

The security features are enabled by compiling with the `-mbranch-protection=standard` gcc command line option.

Note - these security features are only found on newer versions of the AArch64 architecture, and they need a compiler and a loader that will support them. Currently this means Fedora 35 or later, but not RHEL-8 or RHEL-9.

If an assembler source file is used as part of an application then it too needs to be updated. Any location in the source code where an indirect branch or function call can land must now have *BTI* as the first instruction executed. This instruction is safe to use even in code that will not be executed in a BTI-enabled environment as it translates into a no-op instruction if not needed.

In addition the assembler needs a note to indicate that it now supports BTI. This note can be added via including this code snippet in the sources:

```
.pushsection .note.gnu.property, "a"
.align 3
.word 2f - 1f
.word 4f - 3f
.word 5            /* NT_GNU_PROPERTY_TYPE_0 */
1:  .asciz "GNU"

2:  .align 3
3:  .word 0xc0000000   /* type: GNU_PROPERTY_AARCH64_FEATURE_1_AND */
.word 6f - 5f      /* size */
5:  .word 1            /* value: GNU_PROPERTY_AARCH64_FEATURE_1_BTI */

6:  .align 3
4:  .popsection
```

Note - this test is the inverse of the Section 4.2.1.21 [Test not dynamic tags], page 34, test and directly related to the Section 4.2.1.3 [Test branch protection], page 22, test.

Note - this test is automatically enabled if one of the following profile options is used:

```
--profile=rawhide
--profile=f38
--profile=f37
--profile=f36
--profile=el10
```

The test is automatically disabled if one of the other profile options is used, ie:

```
--profile=el7
--profile=el8
--profile=el9
--profile=f35
```

If necessary the test can be disabled via the `--skip-dynamic-tags` option and re-enabled via the `--test-dynamic-tags` option.

4.2.1.7 The entry test

```
  Problem:  Intel's CET security feature requires that the first instruc-
tion in a program be ENDBR
  Fix By:   Compile statup code with -fcf-protection
  Waive If: The application will not run on the latest Intel hardware

  Example:  FAIL: entry test because instruction at entry is not ENDBR32
  Example:  FAIL: entry test because instruction at entry is not ENDBR64
```

This test checks to make sure that the first instruction in a program for the x86_64 architectures is *ENDBR*. This is needed as part of Intel's *CET* security feature. (See Section 4.2.1.4 [Test cf protection], page 24, for more details on *CET*).

If necessary the test can be disabled via the `--skip-entry` option and re-enabled via the `--test-entry` option.

4.2.1.8 The -Ofast test

```
  Problem:  Combining code that is compiled with -Ofast with code that
             is not results in inconsistent behaviour
  Fix By:   Compiling everything with -Ofast
  Waive If: The application does not use maths functions

  Example:  FAIL: fast test because some parts of the program were compiled with -
Ofast and some were not.
```

This test checks that if part of the application was compiled with the `-Ofast` option, then all of it was compiled with this option. If only some parts are compiled with -Ofast then the parts that are not may not behave correctly as they will be expecting to receive accurate results from mathematical functions. For more details on this problem see: `https://bugzilla.redhat.com/show_bug.cgi?id=1248744`

If necessary the test can be disabled via the `--skip-fast` option and re-enabled via the `--test-fast` option.

4.2.1.9 The FIPS test

```
  Problem:  GO binaries need to be compiled with FIPS crypto support
  Fix By:   Compiling CGO_ENABLED=1
  Waive If: The application does not use crypto

  Example:  FAIL: fips test because the binary loaded a non-FIPS compli-
ant crypto library
```

By default when using Fedora, applications written in GO use cryptographic functions from the GO standard library, which is not FIPS-validated. RHEL however is based on upstream GO's dev.boringcrypto branch, which is modified to use BoringSSL for crypto primitives. These are FIPS-validated.

For the best security GO applications should use a FIPS-validated cryptographic library, and this test checks for this behaviour.

This is automatically disabled if a Fedora `profile` is used.

If necessary the test can be disabled via the `--skip-fips` option or re-enabled via the `--test-fips` option.

4.2.1.10 The flex arrays test

```
Problem:  Flexible arrays are a C coding convention that are often
   subject to buffer overrun attacks
Fix By:   Compiling with -fstrict-flex-arrays=[123]
Waive If: The application does not use flexible arrays

Example:  FAIL: flexible test because -fstrict-flex-arrays was not enabled
```

This is a future test. It is not enabled by default. It checks a security feature that may not be widely available or enforced.

This test checks that the application was compiled with the `-fstrict-flex-arrays=[123]` command line option enabled. This option enforces a stricter use of flexible arrays that is easier for the compiler to check for ppotential buffer overrun attacks.

The test also checks that the `-Wstrict-flex-arrays` warning is enabled.

If necessary the test can be disabled via the `--skip-flex-arrays` option and re-enabled via the `--test-flex-arrays` option.

4.2.1.11 The fortify test

```
  Problem:  Buffer overruns in string/memory library functions can be ex-
ploited by an attacker
  Fix By:   Compiling with -D_FORTIFY_SOURCE=2 or -D_FORTIFY_SOURCE=3
  Waive If: The application does not use C library string/memory functions

  Example:  FAIL: fortify test because -D_FORTIFY_SOURCE=[2|3] was not present on the c
mand line
  Example:  FAIL: fortify test because -O level is too low
  Example:  FAIL: fortify test because no indication that the necessary op-
tion was used (and a C compiler was detected)
```

This test checks that the application was compiled with either `-D_FORTIFY_SOURCE=2` or `-D_FORTIFY_SOURCE=3` specified on the compiler command line. Since these options need good optimization in order to work properly the test also checks that `-O2` or higher was used.

The `_FORTIFY_SOURCE` define enables the use of secure version of certain string and memory C library functions. For full details of what it does, see this blog: `https://access.redhat.com/blogs/766093/posts/1976213`

Any program that uses the string or memory functions in the *glibc* library should have this define present on the compiler command line. Programs that do not use these functions do not need the define, but it will not hurt to have it present anyway.

Note - this test is automatically disabled if the `--profile=el7` option is used.

Note - if either of the `el10` or `rawhide` profiles are enabled then only `-D_FORTIFY_SOURCE=3` will be accepted.

If necessary the test can be disabled via the `--skip-fortify` option and re-enabled via the `--test-fortify` option.

4.2.1.12 The gaps test

```
Problem:  Without complete coverage of the compiled code, other tests can miss proble
Fix By:   Compile with -fplugin=annobin
Waive If: Not compiling C or C++ code

Example:  FAIL: gaps were detected in the annobin coverage
```

This test checks to make sure that there are no gaps in the annobin data for the binary being checked. If gaps are present then this means that some parts of the program might have problems which cannot be detected by annocheck.

If necessary the test can be disabled via the `--skip-gaps` option and re-enabled via the `--test-gaps` option.

4.2.1.13 The glibcxx-assertions test

```
  Problem:  Compiled C++ code might contain bugs that could have been de-
tected and fixed
  Fix By:   Compile with -D_GLIBCXX_ASSERTIONS
  Waive If: Not compiling C++
  Waive If: Not using functions from libstdc++

  Example:  FAIL: glibcxx-assertions test because compiled without -D_GLIBCXX_ASSERTION
```

This test checks to make sure that the `-D_GLIBCXX_ASSERTIONS` g+= compiler command line option was used when building binaries. This option is one of several supported by the *libstdc++* library and it is used to enable various NULL pointer and bounds checking security features. For more information see:

`https: / / gcc . gnu . org / onlinedocs / libstdc++ / manual / using_macros.html`

If necessary the test can be disabled via the `--skip-glibcxx-assertions` option and re-enabled via the `--test-glibcxx-assertions` option.

4.2.1.14 The gnu-relro test

```
Problem:  An attacker could alter how an applications interacts with shared libraries
```

```
    Fix By:   Link with -Wl,-z,relro,-z,now
    Waive If: The application runs in an space/time constrained environment

    Example:  FAIL: gnu relro test because not linked with -Wl,-z,relro
```

Some parts of an executable need to be modified when it starts, so that it can access any shared libraries that it uses. This process is called relocation, and once it is finished the altered code/data should not be modified again. The *gnu relro* test checks that write permission can be removed once the relocations have finished.

Enabling *gnu relro* increases the executable size on disk and in memory, and depending upon the application it can also cause a slow start time. It does not cause any significant execution time penalty, and using pre-linking can eliminate the startup penalty.

For programs that do not need to be reloaded often, such as daemons and servers, and on systems where disk and memory are relatively abundant such as desktops and servers, the overhead of *gnu relro* is very insignificant and highly recommended. For programs that need to be reloaded often, the execution penalty of *gnu relro* can be eliminated by using prelinking. For embedded systems where space is scarce, *gnu relro* is not recommended due to its space overhead.

To turn on *gnu relro* compile with the gcc `-Wl,-z,relro,-z,now` option.

If necessary the test can be disabled via the `--skip-gnu-relro` option and re-enabled via the `--test-gnu-relro` option.

4.2.1.15 The gnu-stack test

```
    Problem:  An attacker could place code on the stack and then run it
    Fix By:   Updating compiler, assembler sources and/or linker scripts
    Waive If: The application *really* needs to be able to dynamically cre-
  ate and execute code

    Example:  FAIL: gnu-stack test because the .stack section has incorrect permissions
    Example:  FAIL: gnu-stack test because the .note.GNU-stack section has ex-
  ecute permission
    Example:  FAIL: gnu-stack test because the GNU stack segment has execute permission
    Example:  FAIL: gnu-stack test because the GNU stack segment does not have both read
    Example:  FAIL: gnu-stack test because no .note.GNU-stack section found
    Example:  MAYB: gnu-stack test because multiple stack sections detected
```

This test checks that it is not possible to place code onto the stack and then execute it. Normally the stack just holds data and addresses, but never instructions. A favourite tactic of attackers however is to discover a buffer overrun bug that addresses the stack and then place instructions there before forcing the processor to execute them.

The test actually checks several different parts of a binary file in order to determine that its stack is safe, which is why there are several different potential failure messages.

Most applications will have a section inserted into them by the compiler called *.note.GNU-stack*. The section has no contents, but the read, write, and execute attributes of the section reflect the needs of the application's stack.

Ordinary compiled code should never see this problem, but the test failure can be triggered by programs built with an old compiler which does not support the *.note-GNU-stack* section, or if the program contains some assembler source files or linked with a custom made linker map.

To fix the problem either the compiler needs to be upgraded or the linker map needs to be updated or the assembler sources need to be extended to add the *.note-GNU-stack* section by adding code like this:

```
    .section .note.GNU-stack,"",%progbits
```

If necessary the test can be disabled via the `--skip-gnu-stack` option and re-enabled via the `--test-gnu-stack` option.

4.2.1.16 The go-revision test

```
    Problem:  Using old versions of the GO compiler looses out on security enhacements
    Fix By:   Using a newer GO compiler
    Waive If: No new GO compiler is available

    Example:  FAIL: go-revision test because GO revision must be >= 14
    Example:  FAIL: go-revision test because multiple, different GO version strings found
    Example:  FAIL: go-revision test because no Go compiler revision infor-
  mation found
```

This test checks to see that GO code has been compiled by at least a revision 14 compiler. Earlier versions of the compiler do not have all the bug fixes and security enhancements of later versions.

Note - it is likely that the minimum revision of the GO compiler will be increased in the future.

If necessary the test can be disabled via the `--skip-go-revision` option and re-enabled via the `--test-go-revision` option.

4.2.1.17 The implicit values test

```
    Problem:  Binaries built with implicit types for functions and
              variables might not work in environments where pointers
              and integers are different sizes
    Fix By:   Compiling with the warnings enabled and then fixing the
              warnings
    Waive If: Enabling the warnings would generate false positives

    Example:  FAIL: -Wimplicit-int not enabled
              FAIL: -Wimplicit-function-declaration not enabled
```

This test checks to see that gcc's `-Wimplicit-int` and `-Wimplicit-function-declaration` options have been enabled when the binary built. These options are normally enabled when the `-Wall` option is used (see

Section 4.2.1.40 [Test warnings], page 44), but they can also be enabled and disabled individually.

It is important to enable these warnings and then fix any code that triggers them. Otherwise any code that runs in an environment where a pointer is not the same size as an integer is likely to run into problems where the two are used interchangeably.

If necessary the test can be disabled via the `--skip-implicit-values` option and re-enabled via the `--test-implicit-values` option.

4.2.1.18 The instrumentation test

```
Problem:  Instrumented binaries are bigger and slower than regular binaries
Fix By:   Removing instrumentation options from compiler command line
Waive If: Instrumentation is needed

Example:  WARN: Instrumentation enabled - this is probably a mistake for pro-
duction binaries
```

This test checks to see if any of gcc's instrumentation command line options have been used when the binary built. These options are: `_fsanitize`, `-finstrument-functions`, `-p`, `-pg`, and `-fprofile-arcs`.

If necessary the test can be disabled via the `--skip-instrumentation` option and re-enabled via the `--test-instrumentation` option.

4.2.1.19 The lto test

```
Problem:  When LTO is supported by the compiler, it should be used
Fix By:   Using -flto consistently
Waive If: LTO building is not wanted

Example:  FAIL: no indication of LTO, possibly -fno-lto was used
```

This test checks to see if the `-flto` compiler command line option was used. Using link time optimization produces better code and allows for more compile-time security checks to be run. Hence if it is supported by the compiler it should be used.

Annocheck is currently unable to distinguish between code that has been compiled with the `-fno-lto` option and code that has not been compiled with any `-flto` option, which is why the failure message is so vague.

Note - this test is automatically disabled if the `--profile=el7` or `--profile=el8` option is used.

If necessary the test can be disabled via the `--skip-lto` option and re-enabled via the `--test-lto` option.

4.2.1.20 The not-branch-protection test

```
Problem:  Protecting AArch64 binaries needs newer versions of AArch64 cores
Fix By:   Compile with -mbranch-protection=none
Waive If: Not running on AArch64
Waive If: The application will run on Fedora 35 or later.
```

```
    Waive If: The application will not run on newer AArch64 cores.

    Example:  FAIL: not branch protection test because protection enabled
    Example:  FAIL: not branch protection test because only partially disabled
```

Note - this test is the inverse of the Section 4.2.1.3 [Test branch protection], page 22, test and directly related to the Section 4.2.1.21 [Test not dynamic tags], page 34, test.

This test checks to see if the compile time option to enable the AArch64 branch protection security feature was used. This feature is only supported on newer versions of AArch64 core, and will not work on older cores. Hence this test checks to make sure that the option was not used, or was used but was set to disable the feature.

The security features can be disabled by compiling with the `-mbranch-protection=none` gcc command line option.

Note - this test is automatically disabled if one of the following profile options is used:

```
--profile=rawhide
--profile=f38
--profile=f37
--profile=f36
--profile=el10
```

The test is automatically enabled if one of the other profile options is used, ie:

```
--profile=el7
--profile=el8
--profile=el9
--profile=f35
```

If necessary the test can be disabled via the `--skip-not-branch-protection` option and re-enabled via the `--test-not-branch-protection` option.

4.2.1.21 The not-dynamic-tags test

```
    Problem:  Protecting AArch64 binaries needs newer versions of AArch64 cores
    Fix By:   Compile with -mbranch-protection=off
    Waive If: Not running on AArch64
    Waive If: The application will run on Fedora 35 or later.
    Waive If: The application will not run on newer AArch64 cores.

    Example:  FAIL: not dynamic tags test because BTI_PLT and PAC_PLT flags are present i
  namic tags
    Example:  FAIL: not dynamic tags test because BTI_PLT flag is present in the dy-
  namic tags
    Example:  FAIL: not dynamic tags test because PAC_PLT flag is present in the dy-
  namic tags
```

Note - this test is the inverse of the Section 4.2.1.6 [Test dynamic tags], page 26, test and directly related to the Section 4.2.1.20 [Test not branch protection], page 33, test.

This test checks to see if executable AArch64 binaries have been marked as supporting the *BTI* and *PAC* security features. Such features require the support of the run-time loader in order to work, and this test is intended for environments where this support is missing. (Such as RHEL or pre version-35 Fedora).

The security features can be disabled by compiling with the `-mbranch-protection=none` gcc command line option.

Note - this test is automatically disabled if one of the following profile options is used:

```
--profile=rawhide
--profile=f38
--profile=f37
--profile=f36
--profile=el10
```

The test is automatically enabled if one of the other profile options is used, ie:

```
--profile=el7
--profile=el8
--profile=el9
--profile=f35
```

If necessary the test can be disabled via the `--skip-not-dynamic-tags` option and re-enabled via the `--test-not-dynamic-tags` option.

4.2.1.22 The notes test

```
    Problem:  Lack of annobin notes in a binary means that other tests will not work prop
    Fix By:   Compiling with -fplugin=annobin
    Waive If: The annobin plugin is not available

    Example:  FAIL: notes test because gaps were detected in the annobin coverage
    Example:  MAYB: notes test because not all of the .text section is cov-
  ered by notes
    Example:  FAIL: notes test because annobin notes were not found

    Example:  MAYB: lto test because no indication that LTO was used
    Example:  MAYB: stack-clash test because no notes found regarding this test
    Example:  FAIL: fortify test because no indication that the necessary op-
  tion was used (and a C compiler was detected)
    Example:  FAIL: warnings test because no indication that the necessary op-
  tion was used (and a C compiler was detected)
    Example:  FAIL: stack-realign test because stack realign support is mandatory
    Example:  FAIL: branch-protection test because the -mbranch-protection op-
  tion was not used
```

This test checks that there are annobin notes covering all of the file. Annobin notes are generated by the compiler and describe the security features that have been enabled. The notes contain range information, so that it is possible to determine if all of an application has been covered by the notes, or if there are parts that are missing notes.

If annobin notes are missing from a file then some of the other checks run by the *hardened* checker will not work, which can trigger FAIL or MAYB results for those tests.

Annobin notes are normally produced by a compiler plugin which can be enabled via the `-fplugin=annobin` option for gcc or Clang, and the `-fpass-plugin=annobin` option for LLVM. (Note for pre version-13 of LLVM the `-Xclang -load -Xclang annobin` option should be used instead).

Annobin notes can be generated for assembler sources by using the `-Wa,--generate-missing-build-notes=yes` option. Even better would be to add extra code to the assembler sources to create annobin notes that describe the security features supported by the assembler.

Compiling a simple C program with the `-S -fverbose-asm -fplugin=annobin <security option>` options should provide an example of how to encode an annobin note about <*security option*>.

If necessary the test can be disabled via the `--skip-notes` and `--skip-gaps` options and re-enabled via the `--test-notes` and `--test-gaps` options.

4.2.1.23 The only-go test

```
Problem:  Mixing GO and C is unsafe on x86 platforms
Fix By:   Using a new GO compiler
Waive If: Always

Example:  FAIL: only-go test because combining GO and non-GO object files on x86 sys-
        tems is not safe - it disables CET
```

Note - this test is currently disabled. The GO compiler's lack of support for *CET* is a known issue that cannot be addressed by package maintainers. Hence there is no point in issuing an error message.

This test checks to see if GO and C are being used together in the same program. This is a problem for code that is going to run on x86 architectures as the GO compiler does not support Intel's *CET* technology. (See Section 4.2.1.4 [Test cf protection], page 24, for more details on *CET*). The GO language is inheritantly safer than C, but if the two are mixed, then the C parts will be missing out on the protection offered by *CET*.

If necessary the test can be disabled via the `--skip-only-go` option and re-enabled via the `--test-only-go` option.

4.2.1.24 The openssl-engine test

```
Problem:  The OpenSSL ENGINE API is deprecated in RHEL-10 and later
Fix By:   Update the OpenSSL libraries installed and then rebuild the binary
```

```
    Waive If: The application is not intended for RHEL-10

    Example:  FAIL: OpenSSL binary using the deprecated ENGINE API detected
```

This test checks that the if the application uses the OpenSSL library, it does not make use of the decpreated ENGINE API.

OpenSSL Engines are not FIPS compatible and corresponding API is deprecated since OpenSSL 3.0. The engine functionality we are aware of (PKCS#11, TPM) is covered by providers maintained by Crypto Team now. Feel free to reach crypto team (#crypto) in case of questions.

This test is normally only enabled if the profile is unknown or RHEL-10. If necessary the test can be disabled via the `--skip-openssl-engine` option and enabled via the `--test-openssl-engine` option.

4.2.1.25 The optimization test

```
    Problem:  Insufficient optimization prevents security features from working
    Fix By:   Compiling with -O2
    Waive If: The application does not use string/memory functions

    Example:  FAIL: optimization test because optimization level too low
    Example:  FAIL: optimization test because level too low
    Example:  MAYB: optimization test because no valid notes found regard-
  ing this test
```

This test checks that the application was compiled with sufficient optimization enabled.

The C library security hardening features enabled via the `-D_FORTIFY_SOURCE=2` or `-D_FORTIFY_SOURCE=3` preprocessor command line options will only work properly if the compiler is run at an optimization level of at least `-O2`. Hence this test checks to make sure that this level (or higher) has been used.

Normally the only reason for not using `-O2` or higher is because the application is space sensitive and needs to be compiled with `-Os` or the compilation process is so time intensive that using `-O0` is the only way to obtain reasonable build times.

If necessary the test can be disabled via the `--skip-optimization` option and re-enabled via the `--test-optimization` option.

4.2.1.26 The pic test

```
    Problem:  Static binaries are more vulnerable to attacks
    Fix By:   Compile with -fPIC or -fPIE
    Waive If: Don't.

    Example:  FAIL: pic test because -fpic/-fpie not enabled
```

Programs can be compiled to either load at a fixed address in memory (*static* programs) or at a random address assigned at startup time (*dynamic* programs). Static programs are more vulnerable to exploits because an attacker will know exactly where every part of the program is located. Thus building dynamic executables is recommended.

This test checks that the appropriate compiler option has been used to generate dynamic code. For shared libraries this is the `-fPIE` option should be used. For dynamic executables the `-fPIC` option should be used. Note - there are lower case alternatives of these options (ie `-fpie` and `-fpic`) which can also be used. The difference between the lower case and upper case versions is architecture dependent, but usually the lower case version will only work with smaller programs, wheres the upper case version works for all program sizes.

Note - this check is related to the Section 4.2.1.27 [Test pie], page 38, test. This test checks that the correct compile time option has been used. That test checks that the correct link time option has been used.

If necessary the test can be disabled via the `--skip-pic` option and re-enabled via the `--test-pic` option.

4.2.1.27 The pie test

```
    Problem:  Static binaries are more vulnerable to attacks
    Fix By:   Link with -Wl,-pie
    Waive If: Don't

    Example:  FAIL: pie test because not built with '-Wl,-pie' (gcc/clang) or '-
  buildmode pie' (go)
```

Programs can be compiled to either load at a fixed address in memory (*static* programs) or at a random address assigned at startup time (*dynamic* programs). Static programs are more vulnerable to exploits because an attacker will know exactly where every part of the program is located. Thus building dynamic executables is recommended.

This test checks that the appropriate linker option (`-pie`) has been used to generate dynamic executables. The option is only needed for linking executables, not shared libraries.

Note - this check is related to the Section 4.2.1.26 [Test pic], page 37, test. This test checks that the correct linker option has been used. That test checks that the correct compile time option has been used.

Note - this test is automatically disabled if the `--profile=el7` option is used.

If necessary the test can be disabled via the `--skip-pie` option and re-enabled via the `--test-pie` option.

4.2.1.28 The production test

```
    Problem:  Shipping code generated by an experimental compiler is bad
    Fix By:   Compile with a production ready compiler
    Waive If: The code is never going to be shipped

    Example:  FAIL: production test because a production-ready compiler was not used to b
```

This test checks to make sure that the binary was not produced by an experimental compiler. Experimental compilers can be detected by examining their version information, which will include the string *NOT_FOR_PRODUCTION* or *cross from*.

If necessary the test can be disabled via the `--skip-production` option and re-enabled via the `--test-production` option.

4.2.1.29 The property-note test

```
  Problem:  Badly formed or missing GNU property notes can compromise an ap-
plication at runtime
  Fix By:   Investigate and fix the creation of the notes
  Waive If: Using old tools that do not generate the notes

  Example:  FAIL: property-note test because there is more than one GNU Prop-
erty note
  Example:  FAIL: property-note test because the property note does not have ex-
pected name
  Example:  FAIL: property-note test because the property note data has the wrong size
  Example:  FAIL: property-note test because the note section is present but empty
  Example:  FAIL: property-note test because the property note data has an in-
valid size
  Example:  FAIL: property-note test because the IBT property is not enabled
  Example:  FAIL: property-note test because the SHSTK property is not enabled
  Example:  FAIL: property-note test because unexpected property note type
  Example:  FAIL: property-note test because the BTI property is not enabled
  Example:  FAIL: property-note test because the GNU Property note segment not 8 byte a
  Example:  FAIL: property-note test because there is more than one GNU Prop-
erty note in the note segment
  Example:  FAIL: property-note test because .note.gnu.property section not found (it i
tection support
  Example:  FAIL: property-note test because no .note.gnu.property section = no con-
trol flow information
  Example:  FAIL: property-note test because control flow protection is not enabled
```

GNU property notes are special markers in binary files that provide information about the program to the runtime loader. This information is architecture specific and it often includes details about any security features that were enabled when the program was compiled.

This test checks that the property note is present - if needed for the particular architcture - and that it is properly formatted.

Problems with property notes are usually related to other security options being missing, or the use of assembler source files which do not contain their own instructions for creating property notes.

If necessary the test can be disabled via the `--skip-property-note` option and re-enabled via the `--test-property-note` option.

4.2.1.30 The RHIVOS tests

```
Problem:  Development for the RHIVOS environment requires that some
  extra hardening features be enabled.
```

```
    Fix By:   Follow the requirements for RHIVOS delveopment.
    Waive If: The application is not going to be used in the RHIVOS environment.

    Example:  FAIL: INITFIRST dynamic flag seen
    Example:  FAIL: SONAME includes a directory separator character
    Example:  FAIL: SONAME not the same as the filename
    Example:  FAIL: the DT_AUDIT dynamic tag is present
    Example:  FAIL: the DT_AUXILIARY dynamic tag is present
    Example:  FAIL: the DT_DEPAUDIT dynamic tag is present
    Example:  FAIL: the DT_FILTER dynamic tag is present
    Example:  FAIL: the DT_PREINIT_ARRAY dynamic tag is present
    Example:  FAIL: DT_HASH seen without DT_GNU_HASH
    Example:  FAIL: not linked with -Wl,-z,now
    Example:  FAIL: dlopen/dlclose found in symbol table
    Example:  FAIL: GNU TLS version 1 functions found in symbol table
    Example:  FAIL: LOAD segment with Write and Execute permissions seen
```

Deleopment for the RHIVOS environment requires that some extra hardening measures are applied. This test attempts to check for most of these requirements.

Enabling this test automatically enables the `--test-bind-now`, `--test-gnu-relro`, `--test-gnu-stack` and `--test-rwx-seg` tests.

This test is normally only enabled if the `--profile=rhivos` option is used to select the RHIVOS profile. But it can be enabled independently by the `--test-rhivos` option and disabled via the `--skip-rhivos` option.

4.2.1.31 The run-path test

```
    Problem:  An attacker could cause an application to use a corrupted shared library
    Fix By:   Moving the shared libraries needed to a proper location
    Waive If: The application uses shared libraries held in non-standard locations
    Waive If: The linker does not support --enable-new-dtags

    Example:  FAIL: run-path test because the DT_RPATH/DT_RUNPATH dynamic tag is corrupt
    Example:  MAYB: run-path test because the DT_RPATH/DT_RUNPATH dynamic tag ex-
  ists but is empty
    Example:  FAIL: run-path test because the DT_RPATH/DT_RUNPATH dynamic tag con-
  tains a path that does not start with /usr
    Example:  FAIL: run-path test because the DT_RPATH/DT_RUNPATH dynamic tag has a path
  tains '..'
    Example:  FAIL: run-path test because the DT_RPATH/DT_RUNPATH dynamic tag has $ORI-
  GIN after a non-$ORIGIN path
```

An application that uses shared libraries contains information on how to locate those libraries. This information is a list of directories which should be searched for the libraries. The test checks that the list is secure.

The test actually covers several different aspects, such as all directory paths must be absolute, start with */usr* and must not contain ... If any of these rules are broken then an attacker might be able to exploit the search paths to force the application to load their own, corrupted version of a shared library.

In addition if the `--profile=rawhide` option has been enabled then the presence of the *DT_RPATH* dynamic tag will generate a MAYB result, since in Fedora the *DT_RUNPATH* tag is preferred. (The two tags only differ in when they are evaluated by the program loader). The *DT_RUNPATH* dynamic tag should be generated by default, if it is needed, but in some cases it may be necessary to add the `--enable-new-dtags` option to the linker command line, or the `-Wl,--enable-new-dtags` option if you use gcc to drive the linker.

If necessary the test can be disabled via the `--skip-run-path` option and re-enabled via the `--test-run-path` option.

4.2.1.32 The rwx-seg test

```
Problem:  An attacker could add their own code to an executable
Fix By:   Changing the linker script used to create the binary
Waive If: Don't.

Example:  FAIL: rwx-seg test because segment has Read, Write and eXecute flags set
```

This test checks that the file does not have any segments that are

1. have all three of the *read*, *write* and *execute* permissions.
2. have a non-zero size
3. are resident in memory when the program runs
4. do not have an architecture/OS specific type

Code segments should have read and execute permissions, but they should not be writable as otherwise an attacker can overwrite the code. Data segments should have read permission, and possibly write permission as well, but never execute permission as otherwise an attacker might be able to create their own code in a data area.

The linker will normally never create a binary file with a segment with all three permissions, but it is possible to force it to do so by using a custom linker script. If this flaw is detected then whatever linker script is being used should be corrected to remove the problem.

If necessary the test can be disabled via the `--skip-rwx-seg` option and re-enabled via the `--test-rwx-seg` option.

4.2.1.33 The short-enums test

```
Problem:  Compiler options can change the size of enums
Fix By:   Compile with consistent use of the -fshort-enum option
Waive If: Enums are not passed between different compilation units

Example:  FAIL: short-enum test because both short and long enums supported
```

The `-fshort-enums` gcc compiler option can be used to reduce code size by storing enums in a *short* instead of an *int*. But if the code passes enums between functions compiled in different files then the `-fshort-enums` option must be used consistently or there could be problems.

This test checks that either all files in an application were compiled with the `-fshort-enums` option, or that the option was never used.

If necessary the test can be disabled via the `--skip-short-enums` option and re-enabled via the `--test-short-enums` option.

4.2.1.34 The stack-clash test

```
Problem:  Attackers exploiting stack overrun bugs can gain control of an application
Fix By:   Compiling with -fstack-clash-protection
Waive If: Don't

Example:  FAIL: stack-clash test because -fstack-clash-protection not enabled
```

This test checks that the application has been compiled with stack clash protection enabled (either gcc's `-fstack-clash-protection` or LLVM's `SafeStack` attribute. If this feature is not enabled then an attacker could trick the application into overlapping its heap and stack, allowing them to alter both.

Note - if LTO compilation is enabled then this option needs to be provided both when the object files are built and when they are linked together.

For a full explanation of this topic see these blogs:

`https: / / developers . redhat . com / blog / 2017 / 09 / 25 / stack-clash-mitigation-gcc-background`

`https: / / developers . redhat . com / blog / 2019 / 04 / 30 / stack-clash-mitigation-in-gcc-why-fstack-check-is-not-the-answer`

`https: / / developers . redhat . com / blog / 2020 / 05 / 22 / stack-clash-mitigation-in-gcc-part-3`

Note - this test is automatically disabled if the `--profile=el7` option is used.

If necessary the test can be disabled via the `--skip-stack-clash` option and re-enabled via the `--test-stack-clash` option.

4.2.1.35 The stack-prot test

```
Problem:  Attackers exploiting stack overrun bugs can gain control of an application
Fix By:   Compiling with -fstack-protector-strong
Waive If: Don't

Example:  FAIL: stack-prot test because insufficient protection enabled
Example:  FAIL: stack-prot test because stack protection deliberately disabled
Example:  FAIL: stack-prot test because only some functions protected
Example:  FAIL: stack-prot test because insufficient Stack Safe sanitization
```

This test checks that the application has been compiled with stack protection enabled. For gcc this means using the `-fstack-protector-strong` option and for Clang the `-fsanitize=safe-stack` option. The gcc option does have some levels of protection other than *strong*, but *strong* is the only one that provides full protection.

The stack protection feature adds checks to compiled code that attempt to detect buffer overflows for local buffers. These are often a source of vulnerability that can be exploited by an attacker.

If necessary the test can be disabled via the `--skip-stack-prot` option and re-enabled via the `--test-stack-prot` option.

4.2.1.36 The stack-realign test

```
Problem:  Legacy i686 code is incompatible with SSE instructions
Fix By:   Compile with -mstackrealign
Waive If: The application is not going run in a 32-bit x86 environment
Waive If: The application will not use SSE (or later) instructions

Example:  FAIL: stack-realign test because -mstack-realign not enabled
Example:  FAIL: stack-realign test because stack realign support is mandatory
```

On the Intel 32-bit i686 architecture most instructions work with 4-byte aligned addresses. The *SSE* extension (and later) however need 16-byte aligned addresses. This causes problems for data that is held on the stack, if the stack pointer is not aligned to a 16-byte address. The `-mstackrealign` gcc command line option tells the compiler to generate extra code at function entry which ensures that 16-byte alignment is maintained.

This test checks to make sure that this option has been used when compiling i686 binaries.

If necessary the test can be disabled via the `--skip-stack-realign` option and re-enabled via the `--test-stack-realign` option.

4.2.1.37 The textrel test

```
Problem:  An attacker could change the code in an executable
Fix By:   Compiling with -fPIC enabled
Waive If: The code must be static

Example:  FAIL: textrel test because the DT_TEXTREL tag was detected
```

This test checks to make sure that a binary file does not contain any relocations that alter the contents of a code section. Relocations are special instructions that the program loader uses to alter pieces of a application when it starts up. Normally these relocations are restricted to altering the application's data, but if any of them alter its code then an attacker might be able to exploit this to change the program.

This problem usually only arises when a binary - or part of it - is built to execute at a fixed address. Such binaries need text relocations to help them run at the address chosen. The safest solution therefore is to compile all parts of the binary to be position independent by using the `-fPIC` or `-fPIE` compiler command line options.

If necessary the test can be disabled via the `--skip-textrel` option and re-enabled via the `--test-textrel` option.

4.2.1.38 The threads test

```
Problem:  Programs that do not support exceptions are more vulnerable to attacks
Fix By:   Compile with -fexceptions
Waive If: Program size is an important issue

Example:  FAIL: threads test because not compiled with -fexceptions
```

This test checks to make sure that the `-fexceptions` g++ command line option was used when building the binary. The test is only triggered if the binary uses the *pthreads* library as single threaded applications can cleanly tidy up after themselves if an exception is generated.

If necessary the test can be disabled via the `--skip-threads` option and re-enabled via the `--test-threads` option.

4.2.1.39 The unicode test

```
Problem:  Symbols containing certain unicode characters can conceal their real name
Fix By:   Replacing the unicode characters with other characters
Waive If: The unicode names are valid

Example:  FAIL: unicode test because dangerous characters were found in a sym-
bol name
```

This test checks to make sure that symbols in the binary do not contain control characters or multibyte (aka unicode) characters. Whilst unicode characters are technically allowed in symbol names, their presence is suspect since they can be used maliciously.

The test looks for the following characters in symbol names:

```
Any control character
The space and DEL characters
Any non-unicode multibyte character
```

In addition if the `--test-unicode-all` option has been enabled (either via the command line, or via selecting a RHEL profile with the `--profile` option) then the test will fail is any multibyte character is found.

On the other hand, if the opposite `--test-unicode-suspicious` option has been enabled then the test looks for:

```
Any character with zero width
Any character that changes the direction of the text
```

Other suspicious multibyte characters may be added in the future.

If necessary the test can be disabled via the `--skip-unicode` option and re-enabled via the `--test-unicode` option.

4.2.1.40 The warnings test

```
Problem:  Compiling without warnings enabled can result in poor code
Fix By:   Add -Wall to the compiler command line
Waive If: There are known problems with using -Wall
```

```
  Example:  FAIL: warnings test because compiled without either -Wall or -
Wformat-security
```

This test checks to see that a file has been compiled with either or both of the `-Wall` and `-Wformat-security` options specified. Enabling warnings - and then fixing the problems reported - results in better quality code that is less likely to contain bugs.

If necessary the test can be disabled via the `--skip-warnings` option and re-enabled via the `--test-warnings` option.

4.2.1.41 The writable-got test

```
  Problem:  An attacker could intercept and redirect shared library func-
tion calls
  Fix By:   Link with -Wl,--secure-plt
  Waive If: No shared libraries are used

  Example:  FAIL: writable-got test because the GOT/PLT relocs are writable
```

This test checks that the instructions to set up the *GOT* and *PLT* tables in a dynamic executable cannot be altered by an outside source.

Dynamic executables use two tables to help them connect to shared libraries. These tables - the *GOT* and the *PLT* - are set up when the program runs, based upon instructions held in special sections in the file. If these sections are writable then an attacker could change their contents and thus cause the program to call the wrong functions in the shared libraries.

Under normal circumstances this test should never fail. If it does then something unusual is going on. One possible cure is to add the `-Wl,--secure-plt` option to the final link command line.

If necessary the test can be disabled via the `--skip-writable-got` option and re-enabled via the `--test-writable-got` option.

4.2.1.42 The zero-call-used-regs test

```
  Problem:  An attacker could extract information or use ROP style attacks if call used
isters are not initialised
  Fix By:   Add -fzero-call-used-regs=all
  Waive If: The overhead of initializing the registers is too high

  Example:  FAIL: zero-call-used-regs test because -fzero-call-used-regs not used or se
```

This is a future test. It is not enabled by default. It checks a security feature that may not be widely available or enforced.

This test checks to make sure that programs have been compiled with the `-fzero-call-used-regs=` command line option. This option ensures that registers used in a function call are set to zero when the function returns. If this is not done then a potential attacker might be able to access information in the registers and/or use them in ROP style attacks.

The test can be enabled via the `--test-zero-call-used-regs` option and disabled by the `--skip-zero-call-used-regs` option. It is also enabled if the `--test-future` option is specified and disabled if the `skip-future` option is specified.

4.2.2 Command line options specific to the hardened tool

`--skip-`*`test`*`[=`*`funcname`*`]`
: Disable the test called *test*. If the optional *funcname* argument is supplied then the test is only disabled for the named function (and by implication it is enabled for other functions). This extended version of the option can be used multiple times to allow the test to be skipped for multiple functions.

`--skip-all`
: Disable all tests. Not really useful unless followed by one or more options to enable specific tests.

  Note - using this option also sets the `profile` to *none*. If the enabling of profile specific tests is desired the `--profile` option must appear after the `--skip-all` on the command line.

`--test-`*`name`*
: Enable test *name*.

`--test-all`
: Enable all the tests.

`--test-future`
`--skip-future`
: Report *future fail* tests. These are tests for security features which are not yet implemented or widely adopted, but which are planned for the future. The `--skip-future` option can be used to restore the default behaviour of skipping these tests.

`--test-unicode-all`
`--test-unicode-suspicious`
: The `--test-unicode` test checks for the presence of multibyte characters in symbol names, which are unusual and potentially dangerous. The test has two modes of operation. In one mode, enabled by `--test-unicode-all`, any multibyte character is considered suspicious. This mode is good for code bases where multibyte characters are not expected to appear at all.

  In the other mode, enabled by `--test-unicode-suspicious`, only potentially dangerous unicode characters trigger a failure. See Section 4.2.1.39 [Test unicode], page 44, for more details on which characters are considered suspicious.

  If neither of these options is specified, the default depends upon the profile selected. If a profile is not selected then the default is only fail upon the detection of suspicious characters.

`--profile=el7`
`--profile=rhel-7`
`--profile=el8`
`--profile=rhel-8`
`--profile=el9`
`--profile=rhel-9`
`--profile=el10`
`--profile=rhel-10`
`--profile=rawhide`
`--profile=f40`
`--profile=f39`
`--profile=f38`
`--profile=f37`
`--profile=f36`
`--profile=f35`
`--profile=rhivos`
`--profile=default`
`--profile=none`
`--profile=auto`

Rather than enabling and disabling specific tests a selection can be chosen via a profile option. The `--profile=el7` and `--profile=rhel-7` options will select the tests suitable for *RHEL-7* binaries. Similarly `--profile=el8` or `--profile=rhel-8` configures the tests for *RHEL-8* and so on.

The `--profile=rawhide` option will select tests suitable for *Fedora rawhide* binaries, whilst `--profile=f38` selects tests suitable for *Fedora F38*, and so on for the other Fedora releases.

Other profiles may be added in the future.

The `--profile=rhivos` option enables tests mandated for RHIVOS development.

The `--profile=auto` option will attempt to determine the profile to use, based upon the input filename. This only works with `rpms`, which include the OS as part of their name. This option is the default. The `--profile=default` option is a synonym for the `--profile=auto` option.

Using `--profile=none` will disable the profiling.

For backwards compatibility the form `--profile-<name>` can be used instead of `--profile=<name>`.

Currently the profiles enable and disable the following tests:

`el9`
`f35` Disables the Section 4.2.1.3 [Test branch protection], page 22, and Section 4.2.1.6 [Test dynamic tags], page 26, tests and enables their inverse, ie Section 4.2.1.20 [Test not branch protection],

page 33, and Section 4.2.1.21 [Test not dynamic tags], page 34.

Also enables Section 4.2.1.39 [Test unicode], page 44, and sets the default to fail for any multibyte character.

`el8` Like `el9` but also disables the Section 4.2.1.19 [Test lto], page 33, test.

`el7` Like `el8` but also disables the Section 4.2.1.27 [Test pie], page 38, Section 4.2.1.2 [Test bind now], page 22, Section 4.2.1.11 [Test fortify], page 29, and Section 4.2.1.34 [Test stack clash], page 42, tests.

`el10` Enables the Section 4.2.1.3 [Test branch protection], page 22, and Section 4.2.1.6 [Test dynamic tags], page 26, tests and disables their inverse, ie Section 4.2.1.20 [Test not branch protection], page 33, and Section 4.2.1.21 [Test not dynamic tags], page 34.

`rawhide`
`f36` Like `el10` but also disables the See Section 4.2.1.9 [Test fips], page 28, test.

In addition the Section 4.2.1.39 [Test unicode], page 44, test is enabled for all of the RHEL profiles, but disabled for the Fedora profiles.

`--disable-hardened`
Disable the tool.

`--enable-hardened`
Enable the tool if it was previously disabled. The option is also the default.

`--ignore-gaps`
Do not complain about gaps in the note data.

`--report-gaps`
Do complain about gaps in the note data.

`--fixed-format-messages`
Display messages in a fixed, machine parseable format. The format is:

```
Hardened: <result>: test: <test-name> file: <file-name>
```

Where `<result>` is *PASS* or *FAIL* and `<test-name>` is the name of the test, which is the same as the name used in the `--test-<test-name>` option. The `<filename>` is the name of the input file, but with any special characters replaced so that it always fits on one line.

Here is an example:

```
Hardened: FAIL: test: pie file: a.out.
```

`--disable-colour`
`--enable-colour`
`--disable-color`
`--enable-color`

Do not use colour to enhance FAIL, MAYB and WARN messages. By default annocheck will add colour to these messages so that they stand out when displayed by a terminal emulator. This option can be used in order to turn this feature off. The feature can be re-enabled with `--enable-colour`. The American spelling of color is also supported.

`--full-filenames`
`--base-filenames`

Use the full pathname for files. Useful when recursing into directories. By default this feature is disabled in normal mode and enabled in `verbose` mode. This option and its inverse `--base-filenames` can be used to set a fixed choice.

`--suppress-version-warnings`

Do not issue warning messages about version mismatches between the version of the compiler used to build the annobin plugin and the version of the compiler used to run the annobin plugin.

`--no-urls`
`--provide-urls`

By default when a FAIL or MAYB result is displayed by the *hardened* checker and `--verbose` is enabled, a URL to the online version of the relevant section in this document is also displayed. (Unless the `--fixed-format-messages` option has been enabled). The `--no-urls` option disables the display of the URLs and the `--provide-urls` re-enables the display (even in non-verbose mode).

4.2.3 How to waive the results of the hardening tests

[This section is Red Hat specific.]

Now that `annocheck` is being used by the builders for Fedora and RHEL packages it is possible that certain tests may need to be waived for certain packages. This can be done on a per-package basis by editing the contents of the `rpminspect.yaml` file and adding an entry like this:

```
---
annocheck:
   - hardened: --skip-property-note --ignore-unknown --verbose
```

This example shows how the *property note* test can be ignored. Beware however that doing this overrides the default options that are passed to annocheck by the rpminspect framework, which is why the `--ignore-unknown` and `--verbose` options are also included in the example.

Note - for RHEL the above might not work, as the hardened checker is referred to by another name. So if that appears to be the case, please try:

```
---
annocheck:
   - rhel-policy: --skip-property-note --ignore-unknown --verbose
```

It is also possible to stop annocheck from testing specific files in an rpm by listing them in the `rpminspect.yaml file`, like this:

```
---
annocheck:
    ignore:
        - /usr/libexec/installed-tests/glib/mem-overflow
        - /usr/libexec/installed-tests/glib/resources
```

For more information on rpmdiff see:

`https://docs.engineering.redhat.com/display/HTD/rpmdiff-elf-binarylibrary`

For more information on the use of annobin in RHEL see:

`https://one.redhat.com/rhel-developer-guide/#_annocheck_ensuring_comprehensive_elf_distro_flags`

To get more help on deciding whether or not a test should be waived please ask on either of the *os-devel-list@redhat.com* OS Devel or the *rhel-devel@redhat.com* RHEL Devel mailing lists.

4.2.4 What to do if annocheck reports that it could not find compiled code.

The hardening checker will automatically skip some tests if it cannot prove that the file being checked was created by a known compiler, or if the code was created from assembler sources, rather than a high level language. This is because the test being skipped is checking for a specific feature of a specific compiler.

The checker uses several different methods for determining if an executable was compiled:

`notes`	If there are annobin notes present, these include a description of the compiler used to create the executable.
`DWARF`	If DWARF debug information is available, the compiler can usually be found in the DW_AT_producer tag.
`comment`	If there is a *.comment* section in the file, then this usually contains the name of the compiler.
`GO note`	The presence of a *.note.go.buildid* section indicates that the file contains GO compiler code.
`GO symbol`	The presence of a variable called '`go1.<V>`' in the read-only data section. Again this indicates the presence of GO compiled code.

`executable segments`
: If the file contains one or more program segments with the *executable* flag set, then this indicates that it is likely to contain compiled code.

There are several reasons why annocheck might think that the file does not originate from compiled source code:

`fake assembler`
: Sometimes when compiling code it is desireable to be able to build it without certain security options, but also without annocheck complaining about them. This is used by `glibc` for example because it does not use stack checking or function fortification.

  This effect can be achieved with `GCC` by using the `-Wa,--generate-missing-build-notes` command line option. This tells the assembler to generate a fake annobin note that tells annocheck to treat all of the code as if it has been produced from assembler sources *and* to ignore tests specific to high level languages.

`real assembler`
: The file was created from assembler source code or some other low level language. In this case futher manual checking is warranted. If the source code has not been written with the particular security feature in mind, then it may be vulnerable.

  If on the other hand it does not need the security feature or it has been written to support it, then adding an annobin note to the assembler sources will stop annocheck from complaining.

  For example the following will add a note about strong stack protection:

```
.pushsection .gnu.build.attributes, "o", %note, .text
.balign 4
.dc.l 2f - 1f # Size of the name field.
.dc.l 0 # Empty description field.
.dc.l 0x100 # This is an OPEN note which applies to all the code in the cov
ered region.
      1:
.dc.b 'G', 'A', # This is a GNU attribute
        .dc.b '*'       # It contains a numeric value
        .dc.b 0x2       # For the -fstack-protector option
        .dc.b 0x3       # The value is 3, which indicated -fstack-
protector-strong
        .dc.b 0         # Since this field is nominally a name, it ends wit
      2:
.dc.b 0, 0  # Padding to ensure note ends on a 4 byte boundary.
.popsection
```

Compiling a simple C program with the `-S -fverbose-asm -fplugin=annobin <security option>` options should provide an example of how to encode a note about <*security option*>.

Note - in order for annobin notes to work at least one of them needs to specify the address range that they cover. This is usually done by a version note, which details the version number of the tool used to produce the code. For assembler this note would look like this:

```
.pushsection .gnu.build.attributes, "o", %note, .text
.balign 4
.dc.l 2f - 1f   # Size of the name field.
.dc.l 16 # Size of the description field (= 2 * sizeof (address)).
.dc.l 0x100 # This is an OPEN note which applies to all the code in the cov
ered region.
      1:
.dc.b 'G', 'A', # This is a GNU Attribute note.
        .dc.b '$'       # It contains a string value.
        .dc.b 0x1       # The string is a VERSION string.
        .dc.b '3',      # Version 3 of the Watermark Protocol is be-
ing used.
        .dc.b 'a',      # The code has been produced by an 'a'ssembler.
        .dc.b '2'       # The assembler's major version num-
ber is 2.
        .dc.b 0         # Since this field is nominally a name, it ends wit
      2:
                        # Coincidentally, no padding is needed here.
        .quad <insert-start-symbol-here>
.quad <insert-end-symbol-here>
.popsection
```

`unsupported source language`
: The file was created by compiling high-level language source code, but in a language with which annocheck is unfamiliar. In this case it may still make sense to skip the test, if it is checking for a feature that is not supported by the language's compiler.

`annobin annotation not enabled`
: The file was created by compiling C and/or C++ but without any annobin notation enabled, and without any debug information generation. In this case there may be a problem, since the test being skipped might actually fail if annocheck knew that the file was compiled.

  Fixing this problem involves investigating how the executable was built and adding the necessary rules to invoke the annobin plugin and the sought after security hardening options.

`stripped of symbols and notes`
: The file was built normally, but then stripped of most of the useful information, including its symbol table, debug information, annobin notes and so on.

Restoring the stripped information should solve this problem.

4.3 The annobin note displayer

```
annocheck
  [–disable-hardened]
  –enable-notes
  file...
```

The *notes* tool displays the contents of any annobin notes inside the specified files. It groups the notes by address range, which can help locate missing details.

The *notes* tool is disabled by default, but it can be enabled by the command line option `--enable-notes`. Since the hardening checker is enabled by default it may also be useful to add the `--disable-hardened` option to the command line.

4.4 The section size recorder

```
annocheck
  [–disable-hardened]
  [–size-sec=name]
  [–size-sec-flags=!WAX]
  [–size-seg-flags=!WRX]
  [–size-human]
  [–size-total]
  [–size-missing]
  [–disable-size]
  file...
```

The *section-size* tool records the size of named sections within a list of files and then reports the accumulated size at the end. Since it is part of the `annocheck` framework, it is able to handle directories and rpms files as well as ordinary binary files.

The `--size-sec=name` option enables the tool and tells it to record the size of section *name*. The option can be repeated multiple times to record the sizes of multiple sections. It may also be useful to add the `--disable-hardened` option to the command line as otherwise the security hardening will be run at the same time.

Instead of searching for named sections, it is also possible to search for sections with specific flags. The `--size-sec-flags=<flags>` option will search for any section that has all of the specified <*flags*> set. Currently only *W*, *A* and *X* are recognised as flags, indicating that the section must have the *Write*, *Alloc* or *Execute* flags set respectively. If the *!* exclamation mark character is present then it negates the meaning of the following flags. Thus `--section-sec-flags=W` option will search for any writable section whereas the `--size-sec-flags=W!A` option will search only for sections that are writable but not allocated.

Instead of searching for sections by flags it is also possible to search for segments by flags using the `--size-seg-flags=<flags>` option. The flags recognised for segments are *W* for writable, *R* for readable and *X* for executable. Again the *!* character can be used to invert the meaning of the flags that follow it.

If the `--verbose` option is enabled, then the tool will also report the size of the named section(s) in each file it encounters.

If the `--size-human` option is enabled then sizes will be rounded down to the nearest byte, kibibyte, mebibyte or gibibyte, as appropriate.

If the `--size-total` option is enabled then the total size of all sections in all scanned files will also be reported, as well as the ratio of the reported section sizes to the total section size.

If the `--size-missing` option is enabled then any valid ELF format input file that does not contain any of the sought sections will be reported.

If previously enabled the tool can be disabled via the `--disable-size` command line option.

4.5 How long did the check take ?

```
annocheck
  –enable-timing
  file...
  [–sec]
  [–usec]
  [–nsec]
```

The *timing* tool reports on the time taken by other tools to scan the list of files. The tool is disabled by default, but it can be enabled by the command line option `--enable-timing`.

By default the tool will report times in microseconds, but you can change this to reporting in seconds with the `--sec` or in nanoseconds with the `--nsec`. The default can be restored with the `--usec` option.

5 Allowing other programs to run security checks

The `annocheck` program is mostly seen as a security checking tool and in order to allow third party programs such as `rpminspect` the ability to access these checks a library interface is provided.

An example of how to use the libannocheck library can be found in the annobin testsuite. In particular the `tests/use-libannocheck.c` file contains code to initialise, run and then close the library. In theory however the code flow looks like this:

```
#include <libannocheck.h>
struct libannocheck_internals * handle;
unsigned int num_fails, num_maybs;
handle = libannocheck_init (libannocheck_version, "a.out", NULL);
libannocheck_disable_all_tests (handle);
libannocheck_enable_test (handle, "bind-now");
libannocheck_run_tests (handle, & num_fails, & num_maybs);
libannocheck_finish (handle);
```

The library consists of a header file (`libannocheck.h`) and a shared object file (libannocheck.so). It provides the following functions:

5.1 Initialise the library

```
struct libannocheck_internals *
libannocheck_init (unsigned int  VERSION,
                   const char *  FILEPATH,
                   const char *  DEBUGPATH)
```

Returns a token used to identify the instantiation in future calls.

VERSION is the expected version of the libannocheck library. This should normally be 'libannocheck_version'. If the actual version of the library cannot support VERSION then libannocheck_error_bad_version is returned.

FILEPATH is a path the binary to be tested. It can be absolute or relative. It may not be NULL.

DEBUGPATH is a path the debug info file associated with FILEPATH. It can be NULL.

Returns an enum libannocheck_error cast to a struct libannocheck_internals * if something goes wrong.

5.2 Close the library

```
libannocheck_error
libannocheck_finish (struct libannocheck_internals * HANDLE)
```

Closes the connection to libannocheck. Closes any files opened by the library and releases any memory that is may have allocated. After this any library call using HANDLE should fail.

Returns libannocheck_error_none upon successful closure, otherwise returns an error code.

5.3 Get the library version

```
unsigned int
libannocheck_get_version (void)
```

Returns the actual version number of the libannocheck_library. This should be >= libannocheck_version as defined in the `libannocheck.h` header file.

5.4 Convert an error number into an error message

```
const char *
libannocheck_get_error_message
  (struct libannocheck_internals *  HANDLE,
   enum libannocheck_error          ERRNUM)
```

Returns a (read only) string describing libannocheck error number ERRNUM. Returns NULL if the error code is not recognised.

Handle can be NULL if one is not available. If provided a more detailed error message may be returned.

5.5 Get a list of tests supported by the library

```
libannocheck_error
libannocheck_get_known_tests
  (struct libannocheck_internals * HANDLE,
   libannocheck_test **            TESTS_RETURN,
   unsigned int *                  NUM_TESTS_RETURN)
```

Returns a (read/write) array of tests known to libannocheck in TESTS_RETURN. Returns the number of elements in the array in NUM_TESTS_RETURN. Returns libannocheck_error_none if the retrieval succeeded, or an error result otherwise. The returned array should not be freed.

The array is used by libannocheck internally, so if fields are changed this will affect the library's behaviour. In particular tests can be enabled and disabled without needing to call `libannocheck_enable_test` or `libannocheck_disable_test`.

The test_result_reason and test_result_source fields will initially be NULL. They may have their values changed as a result of a call to `libannocheck_run_tests`.

5.6 Enable all tests

```
libannocheck_error
libannocheck_enable_all_tests (struct libannocheck_internals * HANDLE)
```

Enables all the tests supported by libannocheck.

This function may change some of the fields in the data structure returned by the `libannocheck_get_known_tests` function.

5.7 Disable all tests

```
libannocheck_error
libannocheck_disable_all_tests (struct libannocheck_internals * HANDLE)
```

Disables all of the tests supported by libannocheck. Not normally useful unless followed by code to enable one or more tests.

This function may change some of the fields in the data structure returned by the `libannocheck_get_known_tests` function.

5.8 Enable a specific test

```
libannocheck_error
libannocheck_enable_test
  (struct libannocheck_internals * HANDLE,
   const char *                    TEST_NAME)
```

Enables a specific test. Returns `libannocheck_error_none` upon success or an error code otherwise. If the test is not known then `libannocheck_error_test_not_found` is returned.

This function may change some of the fields in the data structure returned by the `libannocheck_get_known_tests` function.

5.9 Disable a specific test

```
libannocheck_error
libannocheck_disable_test
  (struct libannocheck_internals * HANDLE,
  const char *                     TEST_NAME)
```

Disables a specific test. Returns `libannocheck_error_none` upon success or an error code otherwise. If the test is not known then `libannocheck_error_test_not_found` is returned.

This function may change some of the fields in the data structure returned by the `libannocheck_get_known_tests` function.

5.10 Enable a profile

```
libannocheck_error
libannocheck_enable_profile
  (struct libannocheck_internals * HANDLE,
   const char *                    PROFILE_NAME)
```

Enables and disables certain tests known to be relevant to a specific profile.

Returns libannocheck_error_profile_not_known if the profile is not recognised.

5.11 Get a list of known profiles

```
libannocheck_error
libannocheck_get_known_profiles
  (struct libannocheck_internals * HANDLE,
   const char ***                  PROFILES_RETURN,
   unsigned int *                  NUM_PROFILES_RETURN)
```

Retrieves a (read only) array of profile strings known to libannocheck. The array is returned in PROFILES_RETURN. The number of entries in the array is returned in NUM_PROFILES. Returns libannocheck_error_none upons success, or an error code otherwise.

5.12 Run enabled tests

```
libannocheck_error
libannocheck_run_tests
  (struct libannocheck_internals * HANDLE,
   unsigned int *                  NUM_FAIL_RETURN,
   unsigned int *                  NUM_MAYB_RETURN)
```

Runs all enabled tests.

Returns the number of failed tests in NUM_FAIL_RETURN (if this parameter is not NULL).

Returns the number of "maybe" results in NUM_MAYB_RETURN (if this parameter is not NULL).

Retuns libannocheck_error_none if everything went OK.

Updates the STATE, TEST_RESULT_REASON and TEST_RESULT_SOURCES fields in the entries in the array returned by `libannocheck_get_known_tests` for any enabled test.

Can be called multiple times.

6 Configuring annobin and annocheck

When building annobin and annocheck from the sources there are a few configure options available to customise the build:

`--with-debuginfod`
: debuginfod is a web service that indexes ELF/DWARF debugging resources by build-id and serves them over HTTP.

  By default the `annocheck` program will be built and linked with the debuginfod client library `libdebuginfod` if it is present at build time. The `--with-debuginfod` configure option can be used to force the linking against the library even if the runtime `debuginfod` program cannot be found. Alternatively the `--without debuginfod` can be used to force annobin to be built without `libdebuginfod` support, even if it is present on the build system.

  debuginfod is packaged with elfutils, starting with version 0.178. You can get the latest version from 'https://sourceware.org/elfutils/'.

`--with-gmp=PATH`
: The `--with-gmp=PATH` option can be used to specify an alternative path to the gmp libraries, if necessary.

`--without-libelf`
: The annocheck program uses `libelf` to read ELF binaries. By default the configure system will detect if the library is installed and if not, then it will disable the building of `annocheck` and the running of the tests. (Since they use `annocheck`). This behaviour can be overridden by the `--without-libelf` option which forces the build to assume that libelf is absent even if it would normally be detected.

`--without-tests`
: Disable running the testsuite after building the various binaries.

`--without-clang-plugin`
: Disable the building of the annobin plugin for the Clang compiler.

`--without-llvm-plugin`
: Disable the building of the annobin plugin for the LLVM compiler backend. The LLVM plugin is separate from the Clang plugin and can be used with any language that uses LLVM as a backend compiler.

`--without-gcc-plugin`
: Do not build the gcc plugin.

`--without-docs`
: Do not build the documentation.

`--without-annocheck`
: Do not build the annocheck tool.

`--enable-maintainer-mode`
: This enables the regeneration of the `Makefile` and `configure` files when building the `annobin` sources.

7 How to use the information stored in the binary.

The `annobin` package includes some example scripts that demonstrate how the binary information can be used.

NOTE: These scripts are now redundant, their functionality having been subsumed into the `annocheck` program. However they are still useful as examples of how the annobin data can be consumed, so they are still included in the annobin sources.

The scripts are:

7.1 The built-by script

```
built-by
  [–help]
  [–version]
  [–verbose]
  [–quiet]
  [–silent]
  [–ignore]
  [–readelf=path]
  [–tmpdir=dir]
  [–tool=name]
  [–nottool=name]
  [–before=date]
  [–after=date]
  [–minver=version]
  [–maxver=version]
  [–]
  file...
```

The `built-by` script reports the name and version of the tool used to build the specified file(s). This script also demonstrates how information can be extracted from other other locations in the file, not just the binary annotation notes.

The script can also be used to filter files, only reporting those built by a specific tool, or a specific version of a tool, or even by a version of a tool that was built between a range of dates.

The options available are:

‘`--help`’
‘`-h`’ Displays the usage of the script and then exits.

‘`--version`’
‘`-v`’ Displays the version of the script.

‘`--verbose`’
‘`-V`’ Enables verbose mode, causing the script to detail each action it takes.

'`--quiet`'
'`-q`' Do not include the name of script in the out generated by the script.

'`--silent`'
'`-s`' Produce no output. Just return an exit status.

'`--ignore`'
Do not report file types that do not contain any builder information.

'`--tool=`*name*'
Only report binaries built by *name*. The *name* is only an ordinary string, not a regular expression.

'`--nottool=`*name*'
Skip any binary build by *name*. The *name* is only an ordinary string, not a regular expression.

'`--before=`*date*'
Only report binaries built by a tool that was created before *date*. *date* has the format *YYYYMMDD*.

'`--after=`*date*'
Only report binaries built by a tool that was created after *date*. When combined with the `--before` option can be used to restrict output to files which were built by tools created in a specific date range.

'`--minver=`*version*'
Only report binaries built by a tool whose version is *version* or higher. The *version* string should be in the form *V.V.V*, for example *6.2.1*.

'`--maxver=`*version*'
Only report binaries built by a tool whose version is *version* or lower. Can be combined with the `--minver` option to restrict output to those binaries created by tools within a specific version range.

'`--tmpdir=dir`'
'`-t dir`' Directory to use to store temporary files.

'`--readelf=path`'
'`-r=path`' Use the specified program to read the notes from the files.

'`--`' Stop accumulating command line options. This allows the script to be run on files whose names starts with a dash.

7.2 The check-abi script

```
check-abi
  [–help]
  [–version]
  [–verbose]
  [–quiet]
  [–silent]
  [–inconsistencies]
  [–ignore-unknown]
  [–ignore-ABI|enum|FORTIFY|stack-prot]
  [–readelf=path]
  [–tmpdir=dir]
  [–]
  file...
```

The `check-abi` script reports any potential ABI conflicts in the files specified. This includes the use of the `-fshort-enums` option, the `-fstack-protector` option and the `-D_FORTIFY_SOURCE` option. All of these can affect passing data between functions and hence should be used uniformly throughout the binary.

The script accepts the following command line options:

`--help`
`-h` Displays the usage of the script and then exits.

`--version`
`-v` Displays the version of the script.

`--verbose`
`-V` Enables verbose mode, causing the script to detail each action it takes.

`--quiet`
`-q` Do not include the name of script in the out generated by the script.

`--silent`
`-s` Produce no output. Just return an exit status.

`--inconsistencies`
`-i` Only report files with potential ABI problems.

`--ignore-unknown`
Do not report file types that are not supported or recognised.

`--ignore-`*`ABI|enum|FORTIFY|stack-prot`*
Disables individual ABI checks. Multiple occurrences of this option accumulate. Possible option values are:

‘`ABI`’	Disable checks of the general ABI information.
‘`enum`’	Disable checks of the `-fshort-enum` option.
‘`FORTIFY`’	Disable checks of the ‘`-D_FORTIFY_SOURCE`’ option.

'stack-prot'
: Disable checks of the `-fstack-protect` option.

`--tmpdir=dir`
`-t dir` Directory to use to store temporary files.

`--readelf=path`
`-r=path` Use the specified program to read the notes from the files.

`--` Stop accumulating command line options. This allows the script to be run on files whose names starts with a dash.

7.3 The hardened script

```
hardened
  [--help]
  [--version]
  [--verbose]
  [--quiet]
  [--ignore-unknown]
  [--silent]
  [--vulnerable]
  [--not-hardened]
  [--all]
  [--file-type=auto|lib|exec|obj]
  [--skip=opt|stack|fort|now|relro|pic|operator|clash|cf|cet|realign]
  [--readelf=path]
  [--tmpdir=dir]
  [--]
  file...
```

The `hardened` script reports on the hardening status of the specified file(s). In particular it checks that the whole file was compiled with `-O2` or higher and the `-fstack-protector-strong`, `-D_FORTIFY_SOURCE=2`, `-Wl,-z,now`, `-Wl,-z,relro`, `-fPIE`, `-Wp,-D_GLIBCXX_ASSERTIONS`, `-fstack-clash-protection` `-fcf-protection=full` and `-mcet` options.

The script accepts the following command line options:

`--help`
`-h` Displays the usage of the script and then exits.

`--version`
`-v` Displays the version of the script.

`--verbose`
`-V` Enables verbose mode, causing the script to detail each action it takes.

`--quiet`
`-q` Do not include the name of script in the out generated by the script.

`--ignore-unknown`
`-i` Do not report file types that are not supported or recognised.

`--tmpdir=dir`
`-t dir`
: Directory to use to store temporary files.

`--silent`
`-s`
: Produce no output. Just return an exit status.

`--vulnerable`
`-u`
: Only report files that are known to be vulnerable. Ie files that record all of the necessary information about how they were built, but which were built with an incorrect set of options.

  This option is the default behaviour of the script.

`--not-hardened`
`-n`
: Report any file that cannot be proven to be hardened. This is like the `--vulnerable` option, except that it will also report files that do not record all of the necessary information.

`--all`
`-a`
: Report the hardening status of all of the files examined.

`--file-type=`*`auto|lib|exec|obj`*
`-f=`*`auto|lib|exec|obj`*
: Specifies the type of file being examined. Possible values are:

  ‘`auto`’ Automatically determine the file type from its extension. This is the default.

  ‘`lib`’ Assume all files are shared libraries. Checks that the `-fPIC` option was used.

  ‘`exec`’ Assume all files are executables. Checks that the `-fPIE` option was used.

  ‘`obj`’ Assume all files are object files. Skips checks of the bind now status.

`--skip=`*`opt|stack|fort|now|relro|pic|operator|clash|cf|cet`*
`-k=`*`opt|stack|fort|now|relro|pic|operator|clash|cf|cet`*
: Disables checks of various different hardening features. This option can be repeated multiple times, and the values accumulate. Possible values are:

  ‘`opt`’ Disables checks of the optimization level used.

  ‘`stack`’ Disables checks of the stack protection level.

  ‘`fort`’ Disables checks for `-D_FORTIFY_SOURCE`.

  ‘`now`’ Disables checks for ‘`BIND NOW`’ status.

  ‘`relro`’ Disables checks for ‘`relro`’ or read-only-relocs.

  ‘`pic`’ Disables checks for `-fPIC`/`-fPIE`.

- ‘operator’
  Disables checks for ‘-D_GLIBCXX_ASSERTIONS’.
- ‘clash’ Disables checks for stack clash protection.
- ‘cf’ Disables checks for control flow protection. Note - these checks are only run on x86_64 binaries.
- ‘cet’ Disables checks for control flow enforcement. Note - these checks are only run on x86_64 binaries.
- ‘realign’ Disable checks for stack realignment. Note - these checks are only run on i686 binaries.

--readelf=path
-r=path Use the specified program to read the notes from the files.

-- Stop accumulating command line options. This allows the script to be run on files whose names starts with a dash.

7.4 The run-on-binaries-in script

```
run-on-binaries-in
  [–help]
  [–version]
  [–verbose]
  [–quiet]
  [–ignore]
  [–prefix=‘text’]
  [–tmpdir=dir]
  [–files-from=file]
  [–skip-list=file]
  [–]
  program
  [program-options]
  file...
```

The run-on-binaries-in script allows other scripts, or programs, to be run on the executable files contained inside archives. This includes ‘rpm’ files, ‘tar’ and ‘ar’ files and compressed files.

The script does not recurse into directories, but this can be handled by the find command, like this:

```
find . -type f -exec run-on-binaries-in <script-to-run> {} \;
```

The script accepts the following command line options:

--help
-h Displays the usage of the script and then exits.

--version
-v Displays the version of the script.

--verbose
-V Enables verbose mode, causing the script to detail each action it takes.

If this option is repeated it has the special effect of canceling out the automatic addition of the `-i` to recursive invocations of the script.

`--quiet`
`-q` Do not include the name of script in the out generated by the script.

`--ignore`
`-i` Do not report file types that are not supported or recognised.

This option is automatically enabled when the script is recursively invoked on an archive, unless the `-V -V` has been enabled. This is because it is assumed that archives are likely to contain files that do not need to be scanned.

`--prefix='text'`
`-p 'text'` Add this text to the output from the script when it runs the program on a normal executable.

`--tmpdir=dir`
`-t dir` Directory to use to store temporary files.

`--files-from=file`
`-f=file` Specifies a file containing a list of other files to examine, one per line.

`--skip-list=file`
`-s=file` Specifies a file containing a list of files not to examine, one per line. Blank lines and comments are ignored. Text after a file's name is also ignored. Filenames should start at the beginning of a line.

`--` Stops processing of command line options. This allows the script to be run with a program whose name starts with a dash.

Appendix A GNU Free Documentation License

0. PREAMBLE

   The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

   This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

   We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

   This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

   A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

   A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

ADDENDUM: How to use this License for your documents