Annobin

The ANNOBIN plugin
(Annobin)
Version 10.0

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 objhect 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.

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 `-fpluin=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. The 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 a warning message if it detects that the `-D_FORTIFY_SOURCE=2` has not been provided on the command line and `-flto` has been enabled. This is because LTO compilation hides preprocessor options, so information about them cannot be passed on to the `annocheck` tool.

  The `active-checks` option changes the warning message into an error message, just as if `-Werror` had been specified.

  The `no-active-checks` option disables the warning message entirely.

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

`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.

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 the ELF Note format in a special section called `.gnu.build.attributes`. The `readelf` program from the `binutils` package can extract and display these notes when the `--notes` option is provided. (Adding the `--wide` option is also helpful). 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.

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 and the plugin version will be 9. 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 been disabled. 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.

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.

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
  [–help]
  [–help-tool]
  [–version]
  [–verbose]
  [–quiet]
  [–ignore-unknown]
  [–report-unknown]
  [–debug-rpm=file]
  [–dwarf-dir=dir]
  [–prefix=text]
  [–enable-tool]
  [–disable-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`
: Displays the generic annobin usage information and then exits.

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

`--report-unknown`
`--ignore-unknown`
: These options have two separate effects (and should really be separated into different options). 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 decended.

The second effect is how symbolic links are handled. If reporting is enabled then they are treated as unknown and reported. If reporting is disabled then they are followed, if possible. Otherwise they are reported as being unresolveable.

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, and dangling symbolic links. For other file types, including directories, the default is to report unknown files.

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

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

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

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

`--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-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
  [–disable-hardened]
  –enable-builtby
  [–all]
  [–tool=name]
  [–nottool=name]
  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.

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.

`--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.

4.2 The Hardened security checker.

`annocheck`
[**–skip-all**]
[**–skip-bind-now**]
[**–skip-branch-protection**]
[**–skip-cf-protection**]
[**–skip-dynamic-segment**]
[**–skip-dynamic-tags**]
[**–skip-entry**]
[**–skip-fortify**]
[**–skip-future**]
[**–skip-glibcxx-assertions**]
[**–skip-gnu-relro**]
[**–skip-gnu-stack**]
[**–skip-lto**]
[**–skip-optimization**]
[**–skip-pic**]
[**–skip-pie**]
[**–skip-production**]
[**–skip-property-note**]
[**–skip-run-path**]
[**–skip-rwx-seg**]
[**–skip-short-enum**]
[**–skip-stack-clash**]
[**–skip-stack-prot**]
[**–skip-stack-realign**]
[**–skip-textrel**]

[–**skip-threads**]
[–**skip-warnings**]
[–**skip-writeable-got**]
[–**test-***name*]
[–**test-all**]
[–**test-future**]
[–**profile-el7**]
[–**profile-el9**]
[–**profile-rawhide**]
[–**ignore-gaps**]
[–**fixed-format-messages**]
[–**disable-colour**]
[–**enable-colour**]
[–**disable-hardened**]
[–**enable-hardened**]
[–**full-filenames**]
[–**base-filenames**]
file...

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

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:

4.2.1 The bind-now test

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

4.2.2 The gnu-stack test

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

  Example:  FAIL: the gnu-stack test because the .stack section has incor-
rect permissions
  Example:  FAIL: the gnu-stack test because the .note.GNU-stack section has ex-
ecute permission
  Example:  FAIL: the gnu-stack test because the GNU stack segment has ex-
ecute permission
  Example:  FAIL: the gnu-stack test because the GNU stack segment does not have both r
  Example:  FAIL: the gnu-stack test because no .note.GNU-stack section found
  Example:  MAYB: the 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 attribues of the section reflect the needs of the application's stack.

Ordinary compiled code sholuld never see this problem, but the test failure can be triggered by programs built from assembler sources or linked with a custom made linker map. To fix the problem either the linker map needs to be updated to ensure that the stack section is not executable or the assembler sources need to be extended to a note that the stack is not executable:

```
  .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.

4.2.3 The writeable-got test

```
  Summary:  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 writeable-got test because the GOT/PLT relocs are writeable
```

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 writeable 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 unusal 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-writeable-got` option and re-enabled via the `--test-writeable-got` option.

4.2.4 These tests need extended documentation

`No RWX segments.`
: No program segment should have all three of the read, write and execute permission bits set. Disabled by `--skip-rwx-seg`.

`No text relocations`
: There should be no relocations against executable code. Disabled by `--skip-textrel`.

`Correct runpaths`
: The runpath information used to locate shared libraries at runtime must only include directories rooted at */usr*. Disabled by `--skip-run-path`.

`Missing annobin data`
: The program must have been compiled with annobin notes enabled. Disabled by `--ignore-gaps`.

`Strong stack protection`
: The program must have been compiled with the `-fstack-protector-strong` option enabled, and with `-D_FORTIFY_SOURCE=[2|3]` specified. It must also have been compiled at at least optimization level 2. Disabled by `--skip-stack-prot`.

`Dynamic data present`
: Dynamic executables must have a dynamic segment. Disabled by `--skip-dynamic-segment`.

`Position Independent compilation`
: Shared libraries must have been compiled with `-fPIC` or `-fPIE` but not `-static`. This check can be disabled by `--skip-pic`.

Dynamic executables must have been compiled with `-fPIE` and linked with `-pie`. This check can be disabled by `--skip-pie`.

`Safe exceptions`
: Program which use exception handling must have been compiled with `-fexceptions` enabled and with `-D_GLIBCXX_ASSERTIONS` specified. Disabled by `--skip-threads` and/or `--skip-glibcxx-assertions`.

`Stack Clash protection`
: If available the `-fstack-clash-protection` must have been used. Disabled by `--skip-stack-clash`.

`Control Flow protection`
: If available the `-fcf-protection=full` option must have been used. Disabled by `--skip-cf-protection`. If this option is disabled then the check for GNU Property notes will also be disabled.

`Branch protection`
: For *AArch64* binaries the `-mbranch-protection` option, if available, must have either not. Disabled by `--skip-branch-protection`.

`Stack realignment`
: For *i686* binaries, the `-mstackrealign` option must have been specified. Disabled by `--skip-stack-realign`.

`Source fortification`
: The program must have been compiled with the `-D_FORTIFY_SOURCE=[2|3]` command line option specified. Disabled by `--skip-fortify`.

`Optimization`
: The program must have been compiled with at least `-O2` optimization enabled. Disabled by `--skip-optimization`.

`Link Time Optimization`
: The program must have been compiled with link time optimization (`-flto`) enabled. Currently this is a soft check, so failing this test is not considered a reason to fail the overall run. Disabled by `--skip-lto`.

`Read only relocations`
: The program must not have any relocations that are held in a writeable section. Disabled by `--skip-gnu-relro`.

`GNU Property Note`
: For *x86_64*, *AArch64* and *PowerPC* binaries, check that a correctly formatted GNU Property note is present. The contents of the notes are architecture specific. Disabled by `--skip-property-note`.

`Enum Size` Check that the program makes consistent use of the `-fshort-enum` option.

`Production Ready Compiler`
: Check that the program was built by a production-ready compiler. Disabled by `--skip-production`.

The tool does support a couple of other command line options as well:

`--skip-all`
: Disable all tests. Not really useful unless followed by...

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

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

`--profile-el9`
`--profile-rawhide`
`--profile-el7`
: Rather than enabling and disabling specific tests a selection can be chosen via a profile option. The `--profile-el9` option will select the tests suitable for *RHEL-9* binaries. The `--profile-rawhide` option will select tests suitable for *Fedora rawhide* binaries and the `--profile-el7` option will select tests suitable for *RHEL-7* binaries.

  Other profiles may be added in the future.

`--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.

`--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.

4.2.5 How to waive the results of annocheck tests

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 editting 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. Note 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.

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]
  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 writeable section whereas the `--size-sec-flags=W!A` option will search only for sections that are writeable 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 writeable, *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.

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 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 overrridden 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.

`--with-clang`
: Enable the building of the annobin plugin for the Clang compiler.

`--with-llvm`
: Enable the building of the annobin plugin for the LLVM compiler backend. This 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.

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

6 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:

6.1 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 whoes 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.

6.2 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.

`--inconsitencies`
`-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 occurences 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.

6.3 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.

6.4 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 cancelling 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 whoes name starts with a dash.

Appendix A GNU Free Documentation License

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.

ADDENDUM: How to use this License for your documents